THE CHINESE UNIVERSITY OF HONG KONG

GRADUATION THESIS (TERM 2)

Horse Racing Prediction using Deep Probabilistic Programming with Python and PyTorch (Uber Pyro)

Author:
Yuk WONG

Supervisor:
Prof. Michael R. LYU

LYU1805
Department of Computer Science and Engineering

April 12, 2019

THE CHINESE UNIVERSITY OF HONG KONG

Abstract

Faculty of Engineering

Department of Computer Science and Engineering

BEng degree in Computer Engineering

Horse Racing Prediction using Deep Probabilistic Programming with Python and PyTorch (Uber Pyro)

by Yuk WONG

In this report, we detail the process of applying deep probabilistic programming for horse racing prediction. We investigate the effect of using different set of features for model input. We design a Bayesian neural network model for prediction of the winning horse with a multiple horse representation. Moreover, through repeated experiments, we show that our model can outperform public intelligence and neural networks in terms of both accuracy and net gain. In addition, we demonstrate the effect of different betting strategies on the profitability of our model. Finally, we construct a betting strategy and verified its profitability in the long run with testing data.

Acknowledgements

We would like to express our gratitude to our supervisor Professor Michael R. Lyu and Mr. Edward Lau for providing guidance on data collection, project scoping, and research direction that are essential for completing this project within limited short time frame.

We would also like to thank our friends who helped us a lot on familiarizing machine learning and neural networks as well as setting up the training environment on department server.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Horse racing, sport of running horses at speed, is one of the oldest of all sports and its basic concept has undergone virtually no change over the centuries. In Hong Kong, horse racing is not only a highly developed sport, but also a popular entertainment and gambling game. All betting over horse racing in Hong Kong is regulated and held by the non-profit organization Hong Kong Jockey Club, which holds a legal monopoly and provides different types of bet according to Pari-mutuel betting system. Because of the regulated and transparent betting system, the profitability of horse racing is under active research by statisticians and machine learning specialist alike. However, to the best of our knowledge, there has been no published work obtaining a net profit with neural networks. We attribute this mainly to the variability of horse racing and the insufficiency of training data. The variability of horse racing calls for a complex neural network model in order to model accurately the relations, while the lack of enough training data hinders the training of the large number of parameters associated with a complex model.

One potential way to overcome the shortage of training data is Bayesian inference, a probabilistic technique which has been shown capable of learning from fewer examples [1] [2]. [1] build a Bayesian implementation of learning from just one to five examples by taking advantage of knowledge coming from previously learned categories, no matter how different these categories are might be, while [2] developed a density over transforms shared by many classes and developed a classifier based only a single training example for each class by using the density as prior knowledge. These examples suggest that Bayesian inference may be effective in overcoming the deficit of training data of in our horse racing prediction. However, both works tailor build their models from scratch as one–off systems, limiting their scope and extensibility, and hampering applications of the models to other problems. Due to the time and resources constraints of this project, we do not build our model from zero, instead, we build our model with a probabilistic programming language, Pyro, which provides primitives for sampling and inferring probabilistic distribution, and utilize build–in inference algorithms of Pyro to conduct Bayesian inference.

Probabilistic programming languages unify techniques for the formal description of computation and for the representation and use of probabilistic knowledge [3]. Instead of

programming probabilistic models by hand, probabilistic programming languages provides an abstract means of describing and inferring arbitrary programming models. This enables programmers to build and train large models with less programming efforts.

In this project we build a Bayesian neural network for horse racing prediction with deep probabilistic programming language Pyro. We test different feature selections as well as the different hyperparameters. To demonstrate the performance of our Bayesian neural network, we test two different betting method, fixed betting and Kelly betting. We are able to predict 12 horse races with 22.29% win accuracy and net profit of 7.54%, and 14 horse races with 22.66% win accuracy and net profit of 14.43%.

The rest of the report is structured as follows: Chapter 2 summarizes the past efforts by related works. Chapter 3 describes the motivation of our work. Chapter 4 introduces the background of probabilistic programming, Bayesian inference, and horse racing in Hong Kong. Chapter 5 details our data collection method and preprocessing. Chapter 6 introduces the structure of our model. Chapter 7 describes the implementation of the model. Chapter 8 records the results. Chapter 9 concludes.

Chapter 2 Related Works

Despite the abundant amount of data produced every week in horse racing, few works have been published on the prediction of horse racing. Nonetheless, these few works, utilizing techniques ranging from multinomial logit regression [4] [5] to Support-Vector-Machines [6] and to neural networks [7] [8], have produced motivating results that makes horse racing prediction an attractive topic.

Bolton and Chapman [4] [5] have proposed a multinomial logit modeling approach to handicapping horse races. Sophisticated handicapping factors and a large data base is used to apply a 20-variable pure fundamental multinomial logit model to a 2,000 Hong Kong races, achieving expected returns in excess of 20%. Chung et al. [6] utilized Support-Vector-Machines on a 3-year dataset with 2691 races and 33532 horse records from Hong Kong races. By setting the threshold between highest horse and second highest horse, win accuracy of 35.85%, 56.36%, and 70.86% are achieved at threshold of 0, 0.05, and 0.1, yielding a 840,164.1%, 13692.2%, 2494.8% return respectively.

Cheng and Lau [7], Liu and Wang [8] have used neural networks for horse racing prediction. Cheng and Lau [7] used 16-year dataset of Hong Kong races from 2001 to 2016. Data from 2001 to 2014 are used for training dataset, while data from 2015 to 2016 are used as testing dataset. Their single horse neural network model achieved win accuracy of 21.42%, and when betting on a threshold of 80%, can gain a net profit of 30% in the testing dataset. However, when the threshold is not used, the model result in a loss of over 20%. Liu and Wang [8] used dataset of Hong Kong races from January 2011 to April 2018 with 5029 races and 63459 horse records. Data from 2011 to 2017 are used for training models, while data in 2018 are used as testing data. When betting only on races of specific race classes (Class 1 and Class 2), their best model is able to achieve 48.57% win accuracy and a net gain of 17.45%, but drops to 24.51% and result in a net loss of 25.78% when betting on all races.

Our prior work [9] have used Bayesian neural networks for predicting the place of each individual horse, and achieved 27.96% accuracy and net gain of 39.77% when betting only on specific classes (Class 1 and Group 3), but drops to 25.92% accuracy and net loss of –20.09%.

Chapter 3 Motivation

Past works on applying neural networks for horse racing prediction [7] [8] have not been able to achieve high accuracy. We attribute this mainly to the variability of horse racing and the insufficiency of training data. The variability of horse racing calls for a complex neural network model in order to model accurately the relations, while the lack of enough training data hinders the training of the large number of parameters associated with a complex model. Therefore, this project aims to predict horse racing with Bayesian neural networks which has been shown to able to generalize well from fewer examples [1] [2] compared to other methods.

In addition, past works utilizing neural networks [7] [8] requires additional criteria, such as confidence levels or betting on specific race classes, to generate a profit. The selection of additional criteria after the testing results in information leakage from testing data and therefore cannot be taken as valid result for profitability on unseen data. Thus, this project aims to build a model for end to end prediction of horse racing and generate a profit under all circumstances.

Moreover, it is observed that the although the model of [7] [8] has been able to optimize well in terms of objective function such as mean square error and binary cross entropy, this fails translate to accurate prediction of the winning horse. We attribute this to the fact that the model of [7] [8] only takes one horse into an account during its prediction, thus unable to capture the interaction of different horses in a real race and the small error in individual horses accumulate and reduce the resultant accuracy. In this project, we investigate the effect of employing a multiple horse model and compares its performance with single horse representation of [7] [8].

Chapter 4 Background

4.1 Probabilistic Programming

Probabilistic programs are usual functional or imperative programs, but with two addition properties [10]:

1. The ability to draw values at random from distributions
2. The ability to condition values of variables in a program via observations

However, unlike usual programs which are written for execution, the purpose of probabilistic programs is usually to implicitly infer a probability distribution. For example, probabilistic programs can be used to represent probabilistic graphical models, which use graphs to denote conditional dependences between random variables. The purpose in this example is then to infer the resulting conditional dependences between the unseen variables given the observed subset of random variables.

Compared with non–probabilistic machine learning methods, probabilistic programming techniques has been shown capable of learning from fewer examples [1] [2]. [1] build a Bayesian implementation of learning from just one to five examples by taking advantage of knowledge coming from previously learned categories, no matter how different these categories are might be, while [2] developed a density over transforms shared by many classes and developed a classifier based only a single training example for each class by using the density as prior knowledge. These examples suggest that Bayesian inference may be effective in overcoming the deficit of training data of in our horse racing prediction. However, both works tailor build their models from scratch as one-off systems, limiting their scope and extensibility, and hampering applications of the models to other problems.

The main goal of probabilistic programming languages is to relieve programmers the burden of programming complicate programs for probabilistic sampling and inference which can be represented by only a few mathematical statements [10]. Probabilistic programming languages hides the details of sampling and inference inside the compiler and runtime and enable programmers to express models using their domain expertise and dramatically reduce the programming effort of the programmer for probabilistic modeling. Programmers can then

refocus their effort into developing sophisticated probabilistic models that can accurate model their real–world observations, instead of spending most of their time in programming the details of sampling and inference of their models.

4.2 Bayesian Inference

Bayesian inference gives us a method for learning from data: given a set of latent variables z, we specify a prior distribution $p(z)$ quantifying what we know about before z observing any data; then, we specify how the observed data x relates to z by specifying a likelihood function $p(x|z)$; finally, we apply Bayes' rule

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

to give the posterior function [11]. In the context of probabilistic programming, the model has observations x and latent variables $z$has a joint probability density of the form

$$p(x, z) = p(x|z)p(z)$$

with the following properties:

1. We can sample from each p
2. We can compute the pointwise log pdf p

Bayesian inference is used for probabilistic programs to condition the values of latent variables z via observations, or in other words, inferring the posterior probability $p(z|x)$ given observations x. Applying Bayes' theorem, the posterior probability $p(z|x)$ is calculated by computing the right–hand side of Bayes' theorem. Since $p(x|z)$ can be sampled from forward executions of the model and $p(z)$ is directly defined in the model, only $p(x)$ is remained to be determined for Bayesian inference.

$p(x)$ can be rewritten into the form

$$p(x) = \int p(x|z)p(z)dz$$

but this is difficult to evaluate in general. This is because the integral over the latent variables z often require exponential number of computations for variables of limited possible values, and intractable to calculate if variables z are unbounded with infinite possible values. Therefore, various inference algorithms are developed to conduction Bayesian inference.

4.3 Bayesian Inference Algorithms

Exact Bayesian inference via evaluation of Bayes' theorem is difficult to achieve because calculation of evidence probability $p(x)$ is expensive or even intractable. In this section, we describe two main family of algorithms for Bayesian inference, Markov Chain Monte Carlo algorithms and Variational Inference, the first approximates the evidence probability $p(x)$ via sampling techniques, while the latter approximates the posterior probability $p(z|x)$ by introducing a parameterized distribution serving the approximation to the posterior. In addition, we discuss an algorithm to solve the evidence probability $p(x)$ in the special case where the latent variables z of the model have limited possible values.

4.3.1 Enumeration

In the special case that latent variables z of the model have limited possible values, we can enumerate all the possible values of z and sample the corresponding $p(x|z)$ to obtain $p(x)$.

In this way, $p(x)$ are determined by

$$p(x) = \sum_{z} p(x|z)p(z)$$

and the conditioned value (probability density) of the latent variables z is given by

$$p(z|x) = \frac{p(x|z)p(z)}{\sum_z p(x|z)p(z)}$$

4.3.2 Markov Chain Monte Carlo Algorithms

Markov Chain Monte Carlo algorithms estimate a distribution by first taking a sample z_0 from initial distribution $q(z_0)$, then iteratively sample z_i from the transitional distribution based the previous sample, and by judiciously choosing the transitional distribution, the outcome of this procedure with be a random procedure that converges in distribution to the exact posterior $p(z|x)$ [11]. The basic idea for MCMC algorithms tend to follow the following framework:

1. Sample z_0 from the initial distribution $q(z_0)$
2. Propose a new sample z_i'
3. Accept or reject probabilistically using the $q(z_i|z_{i-1})$ and $p(x|z)$
4. If the proposal is accepted, return to step 2 with z_i

5. If the proposal is rejected, return to step 2 with z_{i-1}
6. After specified number of iterations, return all z_0 to z_{n-1}

The main difference between MCMC algorithms is how the new sample is proposed and how the proposal is decided for acceptance. Here, we introduce one of the simpler Markov Chain Monte Carlo Algorithms: Metropolis Algorithm [12]. The Metropolis algorithm uses a normal distribution to propose a jump. This normal distribution has a mean value μ which is equal to the current position and takes a parameter known as proposal width for its standard deviation σ. The proposal width is a parameter of the Metropolis algorithm and has a significant impact on convergence. A larger proposal width will jump further and cover more space in the posterior distribution but might miss a region of higher probability initially. However, a smaller proposal width won't cover as much of the space as quickly and thus could take longer to converge.

Once new state has been proposed, we need to decide (in a probabilistic manner) whether it is a good move to jump to the new position. In Metropolis algorithm, the ratio of the proposal distribution of the new state and the proposal distribution of the current state is used as the probability of accepting proposal, p:

$$p = \frac{p(z'|x)}{p(z|x)} = \frac{p(x|z')p(z')}{p(x|z)p(z)}$$

Therefore, we are visiting regions of high posterior probability $p(z|x)$ relatively more often that those of low posterior probability.

4.3.3 Variational Inference

Variational inference provides a different approach to approximate Bayesian inference. The basic idea of variational inference is to introduce a parameterized variational distribution $q(z)$ to approximate the posterior $p(z|x)$ [13] [14]. To make the approximate as close as possible to the actual posterior, we search over the space of approximating distributions to find the particular distribution with the minimum Kuller–Leibler divergence with the actual posterior. Then, the problem of inferring the posterior probability is transformed to an optimization problem minimizing the Kuller–Leibler divergence

$$\mathrm{KL}\big(q(z)||p(z|x)\big) = \mathrm{E}_q\left[\log\frac{q(z)}{p(z|x)}\right] = \int q(z)\log\frac{q(z)}{p(z|x)}\,dz$$

which is the measure of the divergence of one probability distribution from a second, reference probability distribution [15]. This measure goes to zero when the approximation $q(z)$ perfectly matches $p(z|x)$.However, since $p(z|x)$ is intractable to compute, we cannot minimize the Kuller–Leibler divergence exactly. A simple derivation from [16] yields an alternative representation allowing minimization

$$\begin{aligned}
\mathrm{KL}\big(q(z)||p(z|x)\big) &= \mathrm{E}_q\left[\log\frac{q(z)}{p(z|x)}\right] \\
&= \mathrm{E}_q[\log q(z)] - \mathrm{E}_q[\log p(z|x)] \\
&= \mathrm{E}_q[\log q(z)] - \mathrm{E}_q\left[\log\frac{p(x,z)}{p(x)}\right] \\
&= \mathrm{E}_q[\log q(z)] - \mathrm{E}_q[\log p(x,z)] + \log p(x) \\
&= -\big(\mathrm{E}_q[\log p(x,z)] - \mathrm{E}_q[\log q(z)]\big) + \log p(x)
\end{aligned}$$

The term inside the bracket is known as Evidence Lower Bound

$$\mathrm{ELBO} = \mathrm{E}_q[\log p(x,z)] - \mathrm{E}_q[\log q(z)]$$

which can be directly maximized because both $p(x,z)$ and $q(z)$ can be computed efficiently.

Notice that $\log p(x)$ does not depend on q, therefore, minimizing the Kuller–Leibler divergence in the parameter space of $q(z)$ is the same as maximizing the ELBO, which can be done via gradient ascent [13]. Let ϕ be the parameters that defines distribution $q(z)$, then, each step of variation inference that maximize ELBO at the learning rate of α are as follows:

1. Calculate ELBO(x, z, ϕ)
2. Calculate $\delta_\phi = \frac{\partial \mathrm{ELBO}}{\partial \phi}$
3. Update $\phi \leftarrow \phi + \alpha\delta_\phi$

4.4 Artificial Neural Networks

Artificial neural networks are collections of connected units or nodes called artificial neurons inspired by the biological neural networks in animal brains [17] [18]. The figure below shows the structure of a unit in neural networks:

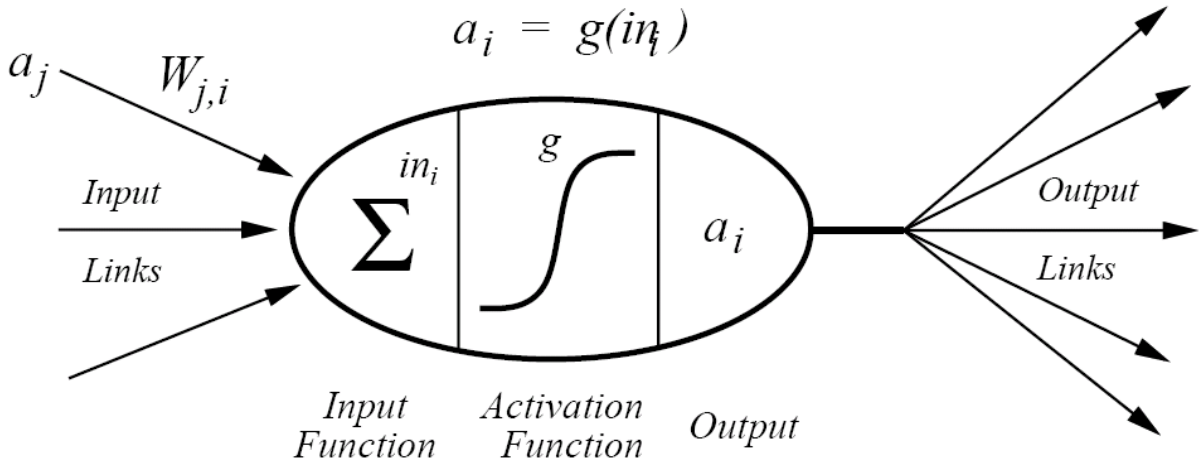


Figure 1. A unit in neural networks, taken from [19]

Each unit in neural networks are composed of an input function, a linear component that computes the weighted sum of the unit's input values, and an activation function, a nonlinear activation component that transforms the weighted sum into a final value serving as the unit's activation value (output value) [19]. The choice of activation functions is different for different network, but the sigmoid function, hyperbolic tangent (tanh), and rectified linear unit (ReLU) are the most popular activation functions. For example, let a_j be the inputs of the unit i, $W_{j,i}$ be the weights of the unit, g be the activation function. The input function computes

$$in_i = \sum_j W_{j,i} a_j$$

, and the activation function transforms the weighted sum into the final output value

$$a_i = g(in_i) = g(\sum_j W_{j,i} a_j)$$

Despite the simple structure of neural networks, it has been found that multilayer feedforward networks can approximate any functions [20]. A 2–layer neural network with 1 hidden layer can approximate any continuous functions, while 3 or more layers neural network with 2 or more hidden layers can approximate any functions. This makes neural networks very useful for modeling unknown relations.

4.4.1 Neural Network Training

Neural networks are commonly trained using gradient descent, which calculates the partial derivative of the error with respect to each parameter by chain rule and adjust the parameter by the product the learning rate and the derivative [19]. For example, let E denote the error, $W_{j,i}$ denotes the weight matrix of the unit to adjust, g be the activation function, a_j be the inputs of the unit, and α denote the learning rate. If the unit is located at the final layer, then the gradient of the weighted sum in_i can be calculated directly

$$\delta_i = \frac{\partial E}{\partial\, in_i} = g'(in_i) \times E$$

and the gradient of each weights

$$\delta_{j,i} = \frac{\partial E}{\partial W_{j,i}} = a_j \frac{\partial E}{\partial\, in_i} = a_j \times \delta_i = a_j \times g'(in_i) \times E$$

Then, each weight is updated by

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times \delta_{j,i}$$

However, if the unit is not located at the final layer, the gradient needs to be calculated by backpropagation, which the error in latter layer i are propagated to earlier layer j:

$$\delta_j = g'(in_j) \sum_i W_{j,i}\, \delta_i$$

and the gradient of each weights

$$\delta_{k,j} = \frac{\partial E}{\partial W_{k,j}} = a_k \frac{\partial E}{\partial\, in_j} = a_k \times \delta_j = a_k \times g'(in_j) \sum_i W_{j,i}\, \delta_i$$

Then, each weight is updated by

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times \delta_{k,j}$$

With gradient descent and backpropagation, each parameter of the neural network can be updated accordingly in each step based on the error after one inference. The whole process is repeated until the network converges [19].

4.5 Deep Probabilistic Programming

Deep probabilistic programming combines neural networks with probabilistic models [21] [22]. By combining neural networks with probabilistic models, deep probabilistic programming is capable of handling hierarchical representation learning while accounting for uncertainty. There are many kinds of different models where neural networks are embedded in probabilistic models, for example, neural networks can be used as for modeling the probabilistic relations between latent variables and observations. The most used deep probabilistic programming models are Bayesian neural networks.

4.5.1 Bayesian Neural Networks

Bayesian neural networks are artificial neural networks inferred by Bayesian inference [23]. Compared to traditional neural networks trained using back propagation, Bayesian neural networks are trained with a distribution instead of a single value for each parameter in the neural networks, such as weights and biases. In context of previous sections, the parameters of the Bayesian neural networks are then latent variables z of a probabilistic model, and the data points are the observations x. Applying Bayesian inference, parameters of Bayesian neural network can be obtained by Bayes' theorem

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

Here, we encounter the same problem of intractable $p(x)$, which calls for the various Bayesian inference algorithms to be applied to infer the parameters of neural networks. For example, consider a Bayesian neural network with prior on the weights and biases z to be the unitary normal and trained using Variational Inference. The prior on the parameters can be expressed as

$$p(z) = \text{Normal}(0,1)$$

Applying Variation Inference, we introduce the variational distribution $q(z)$ to approximate the posterior distribution $p(z|x)$. In this example, we let the variational distribution to be a normal distribution with learnable parameters μ, ρ. To avoid negative values of standard deviation, a softplus function is placed over ρ.

$$q(z) = \text{Normal}(\mu, \log(1 + e^{\rho}))$$

Then, each step of variation inference at the learning rate of α are as follows:

1. Sample ϵ from $\mathrm{Normal}(0,1)$
2. Sampled $z = \mu + \epsilon \log(1 + e^{\rho})$
3. Calculate ELBO(x, z, μ, ρ)
4. $\delta_{\mu} = \frac{\partial ELBO}{\partial z} + \frac{\partial ELBO}{\partial \mu}$
5. $\delta_{\rho} = \frac{\partial ELBO}{\partial z} \frac{\epsilon}{1+e^{-\rho}} + \frac{\partial ELBO}{\partial \rho}$
6. Update $\mu \leftarrow \mu + \alpha\delta_{\mu}$, $\rho \leftarrow \rho + \alpha\delta_{\rho}$

Note that $\frac{\partial ELBO}{\partial z}$ can be obtained from the backpropagation of standard neural networks.

4.6 Horse Racing

Horse racing, sport of running horses at speed, is one of the oldest of all sports and its basic concept has undergone virtually no change over the centuries. In Hong Kong, horse racing is not only a highly developed sport, but also a popular entertainment and gambling game. All betting over horse racing in Hong Kong is regulated and held by the non–profit organization Hong Kong Jockey Club, which holds a legal monopoly and provides different types of bet according to Pari-mutuel betting system.

Pari-mutuel betting is a betting system in which the stake of a particular bet type is place together in a pool, and the returns are calculated based on the pool among all winning bets [24] Dividend will be shared by the number of winning combinations of a particular pool. Winners will share the percentage of pool payout in proportion to their winning stakes. The following tables taken from [24] show the different betting types:

Single–race Pool	Dividend Qualification
Win	1st in a race
Place	1st, 2nd or 3rd in a race, or 1st or 2nd in a race of 4 to 6 declared starters
Quinella	1st and 2nd in any order in a race
Quinella Place	Any two of the first three placed horses in any order in a race
3 Pick 1 (Composite Win) **Winning Trainer (Composite Win)** **Winning Region (Composite Win)**	Composite containing the 1st horse in a race
Tierce	1st , 2nd and 3rd in correct order in a race
Trio	1st, 2nd and 3rd in any order in a race
First 4	1st, 2nd , 3rd and 4th in any order in a race
Quartet	1st, 2nd , 3rd and 4th in correct order in a race

Table 1. Types of bets in Single–race Pool

Multi–race Pool	Dividend Qualification
Double	1st in each of the two nominated races
	Consolation :1st in 1st nominated race and 2nd in 2nd nominated race
Treble	1st in each of the three nominated races
	Consolation : 1st in the first two Legs and 2nd in 3rd Leg of the three nominated races

Table 2. Types of bets in Multi–race Pool

Jackpot Pool	Dividend Qualification
Double Trio	1st, 2nd and 3rd in any order in each of the two nominated races
Triple Trio	1st, 2nd and 3rd in any order in each of the three nominated races
	Consolation : Select correctly the 1st, 2nd and 3rd horses in any order in the first two Legs of the three nominated races
Six Up	1st or 2nd in each of the six nominated races
	Six Win Bonus :1st in each of the six nominated races

Table 3. Types of bets in Jackpot Pool

Among these pools, Hong Kong Jockey Club does not pay out 100% of the pool amount. Instead, the percentage of the pool payout are according to the following tables from [24]:

Single Pool	Percentage of Pool Payout
Win / Place / Quinella / Quinella Place / Double	82.5%
Tierce / Trio / First 4 / Quartet / Double Trio / Triple Trio / Treble / Six Up	75%

Table 4. Percentage of Pool Payout in Single Pool

Merged Pool	Percentage of Pool Payout
Win (including Composite Win)	82.5%
Quartet & First 4	75%

Table 5. Percentage of Pool Payout in Merged Pool

Because Hong Kong Jockey Club does not pay out 100% of the pool amount, there is an expected loss of 17.5% for bets Win, Place, Quinella, Quinella Place, and Double bets, and an expected loss of 25% for Tierce, Trio, First 4, Quartet, Double Trio, Triple Trio, Treble, Six Up. The expected loss makes generating profit from horse racing a challenging task. In a Pari-mutuel pool with 100% payout, an individual only has to slightly beat the public intelligence to have a net gain, but to generate a profit from a Pari-mutuel pool with only 75% or 82.5% payout, an individual needs to beat the public intelligence by a large amount to generate profit.

Chapter 5 Data

5.1 Data Collection

Many companies sell horse racing data online. One possible way to obtain training data for our model is to purchase from them. However, due to the lack of budget and the questionable authenticity of these data, we decided to collect the data from the official website of Hong Kong Jockey Club.

5.2 Data Description

The horse racing dataset contains 8 years of racing data from January 1, 2011 to December 31, 2018. Each entry in the dataset represent the information of a horse in a race. The dataset contains 77562 records from 6251 races taken place in Hong Kong. In addition to race data, we also scrape corresponding horse data from HKJC website and weather data from TimeAndDate. Apart from the obtaining raw data, we also add some features extracted from the data.

The dataset is split into two parts: training dataset and testing dataset. The training dataset contains data from 2011 to 2017 with 68074 records and 5461 races, while the testing dataset contains data of the year 2018 with 9489 records and 790 races.

The following tables describes the races features, horse features obtained from HKJC website, weather features obtained from TimeAndDate, and additional features we extracted.

Feature	Description	Types	Values
raceyear	Year of the race	Date	–
racemonth	Month of the race	Date	–
raceday	Day of the race	Date	–
raceid	Unique id of the race	Index	–
location	Location of the race	Categorical	ST, HV
class	Class of the horses	Categorical	Class 1 to 5, Group 1 to 3
distance	Distance of the race	Categorical	1000, 1200, 1400, 1600, 1650, 1800, 2000, 2200, 2400
course	Track used for the race	Categorical	A, A+3, AWT, B, B+2, C, C+3
going	Soil measurement	Categorical	FIRM, GOOD TO FIRM, GOOD, GOOD TO YIELDING, YIELDING, YIELDING TO SOFT, FAST, SLOW, WET FAST, WET SLOW
raceno	Race number in a race day	Categorical	1 to 11
horseno	Number assigned by HKJC to horse	Categorical	1 to 14
horseid	Unique id of horse	Categorical	4373 distinct values
jockeycode	Unique id of jockey	Categorical	169 distinct values
trainercode	Unique id of trainer	Categorical	147 distinct values
draw	Draw of the horse in race	Categorical	1 to 14
actualweight	Weight added to horse	Real value	–
horseweight	Weight of horse itself	Real value	–
winodds	Bet return on win	Real value	1 to 99
place	Place of horse in race	Categorical	1 to 14
finishtime	Finishing time of horse	Real value	–

Table 6. Race features from HKJC website

Feature	Description	Types	Values
origin	Place of origin	Categorical	
age	Age of horse	Real value	3 to 10
color	Color of horse	Categorical	Bay, Black, Brown, Chestnut, Dark, Grey, Roan
sex	Sex of horse	Categorical	Colt, Filly, Gelding, Horse, Mare, Rig
sire	Father of horse	Categorical	786 distinct values
dam	Mother of horse	Categorical	3786 distinct values
dam's sire	Maternal grandfather of horse	Categorical	1009 distinct values
horseid	Unique id of horse	Categorical	4373 distinct values

Table 7. Horse features from HKJC website

Feature	Description	Types	Values
location	Location of the race	Categorical	ST, HV
temperature	Air temperature	Real value	–
weather	Description of weather	Categorical	Bay, Black, Brown, Chestnut, Dark, Grey, Roan
wind _speed	Wind speed in km/h	Real value	–
wind_direction	Wind direction	Categorical	–
humidity	Humidity	Real value	0 to 100
moon	Moon phase	Real value	0 to 29.5305882
raceid	Unique id of the race	Index	–

Table 8. Weather features from TimeAndDate

Feature	Description	Types	Values
dn	Day or Night	Categorical	D, N
old_place	Place of horse in last race	Categorical	1 to 14
weightdiff	Difference in weight from previous race	Real value	–

Table 9. Extracted features

5.3 Features Analysis

Some features, like origin, age, color, and sex of the horse are traditional considered as important factors in determining the wining horse. In this section, we investigate the effect of these features on the winning horse.

5.3.1 Origin

Historically, the best performing horses comes from Britain, Ireland, and the United States, but recently some of the best horses come from Australia and New Zealand [25]. In addition, the guiding principle for breeding winning racehorses has always been best expressed as "breed the best to the best and hope for the best" [26]. Therefore, the origin of the horse may have an impact on the horse performance.

To analyze whether the origin of the horse is really correlated to the winning probability, we have plotted the origin distribution of winning horse on the next page.

Note that the origin distribution of winning horse alone is not enough to determine the winning probability of horses of different origin, since the distribution is obscured by the origin distribution of all horses. Therefore, we also plot the conditional origin distribution of winning horses.

From the figure, it can be inferred that the origin of the horses has an influence over the winning probability with horses from Japan and Australia having the highest winning probability and horses from Brazil and Spain having the lowest winning probability. Therefore, the origin of the horses should be included for the model input.

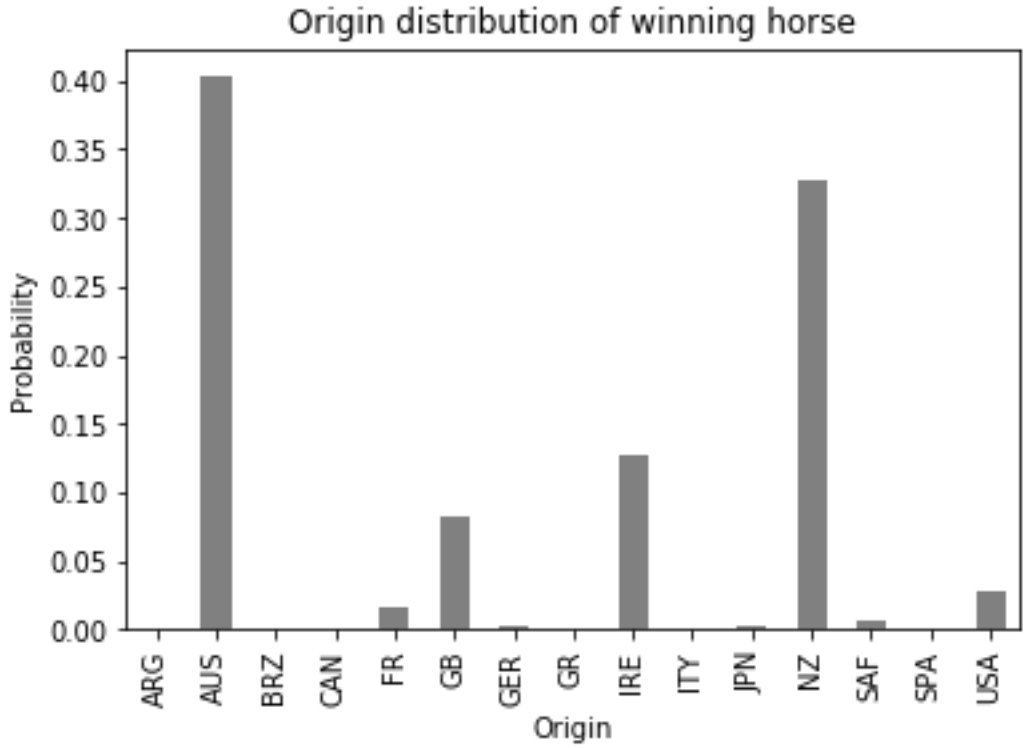


Figure 2. Origin distribution of winning horse

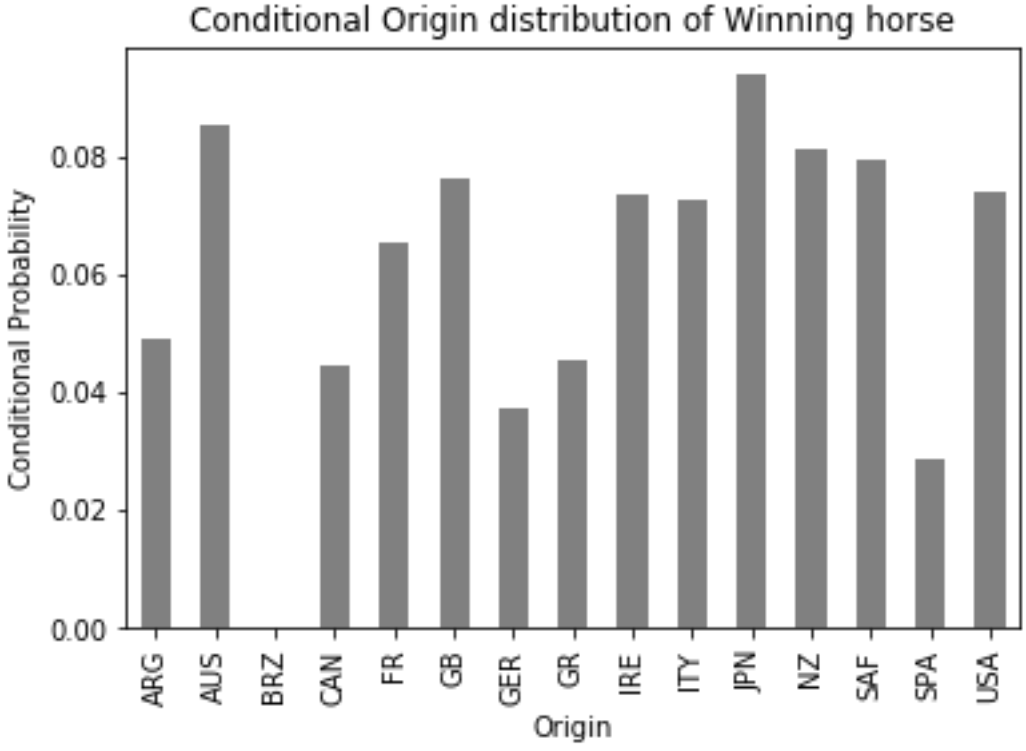


Figure 3. Condition Origin distribution of winning horse

5.3.2 Age

The racing career of the horse is from age 2 to 10 and retirement is mandatory at age 11. The age of the horse is directly related to its performance. Usually, horses reach their peak performance at age 4 to 6 [27], and start to age subsequently and decrease in performance.

To verify whether the statement above is true, the age distribution and the conditional age distribution of the winning horse is shown on the next page.

From the figure, it can be inferred that horse's age has a large influence on performance, with age 2 horses having the highest winning probability and age 4 having the second highest winning probability. Therefore, age is an important feature for predicting the winning horse and should be included for the model input.

In addition, it should be noted that age 2 horses are not common and only contribute to a small number of wins, which may because these age 2 horses are prodigies with exceptional performance. Other horses join horse racing at age 3 and takes a year to gain experience and reach peak performance.

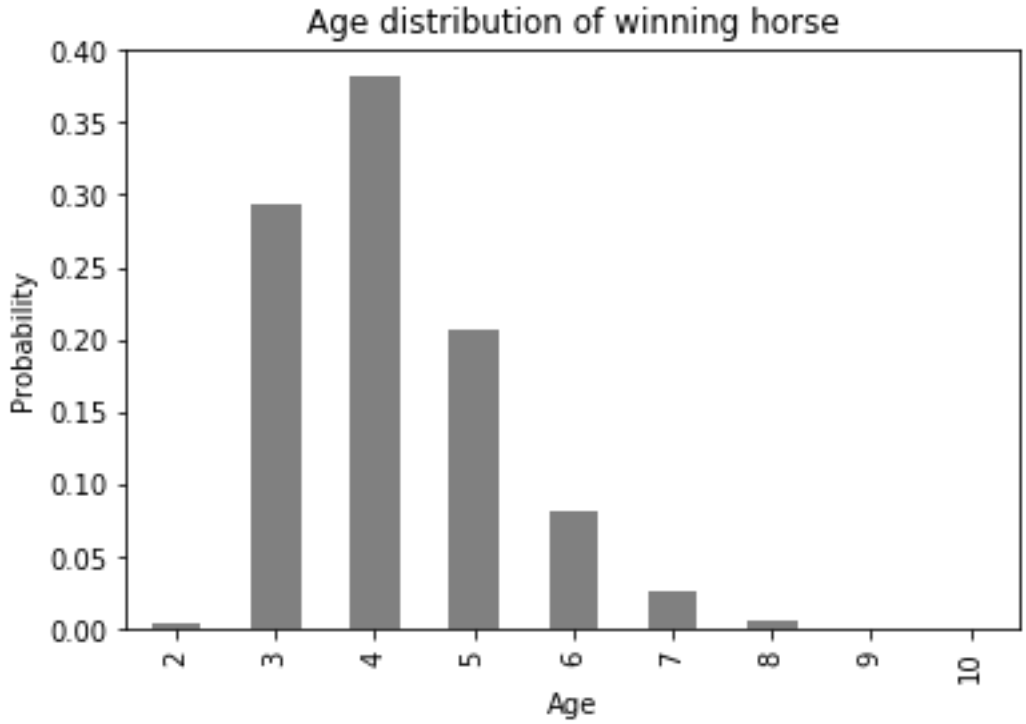


Figure 4. Age distribution of winning horse

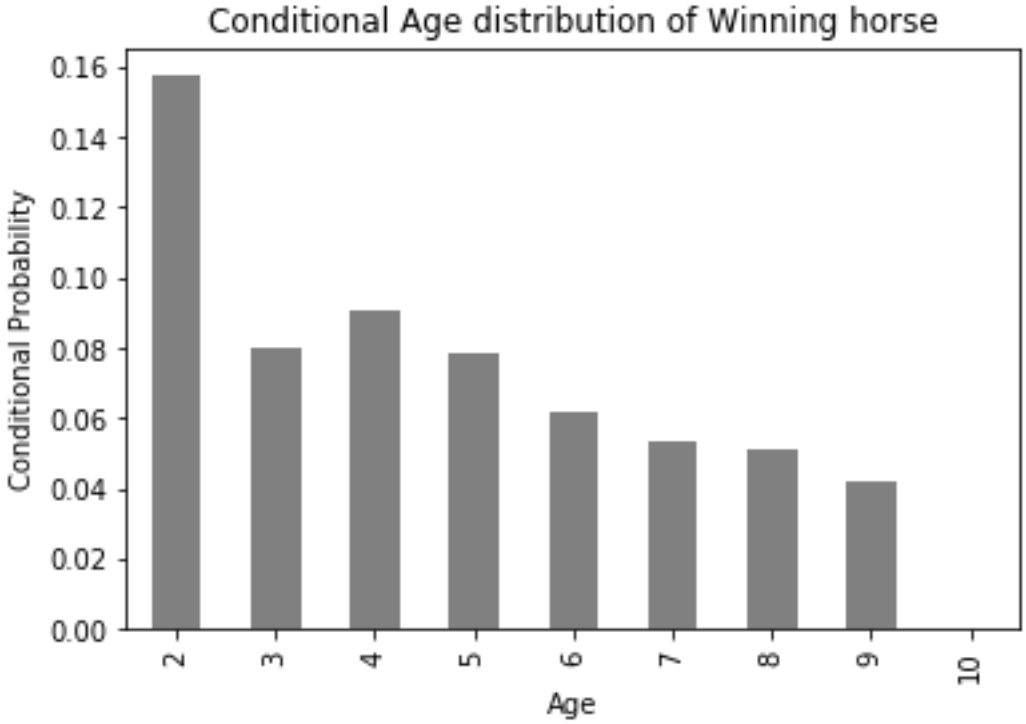


Figure 5. Conditional Age distribution of winning horse

5.3.3 Color

It is commonly believed that the color of the horse indicates the horse's performance. In Hong Kong, the major types of colors are Chestnut, Brown, Bay and Grey [27]. To analyze whether color is a factor correlated to winning probability, the color distribution and the conditional color distribution of the winning horse is shown on the next page.

From the figure, it can be inferred that color is correlated to winning probability with horses of dark and roan color being more likely to win while horses of bay, black, brown, and chestnut have similar winning probability. Therefore, color should be included for model input.

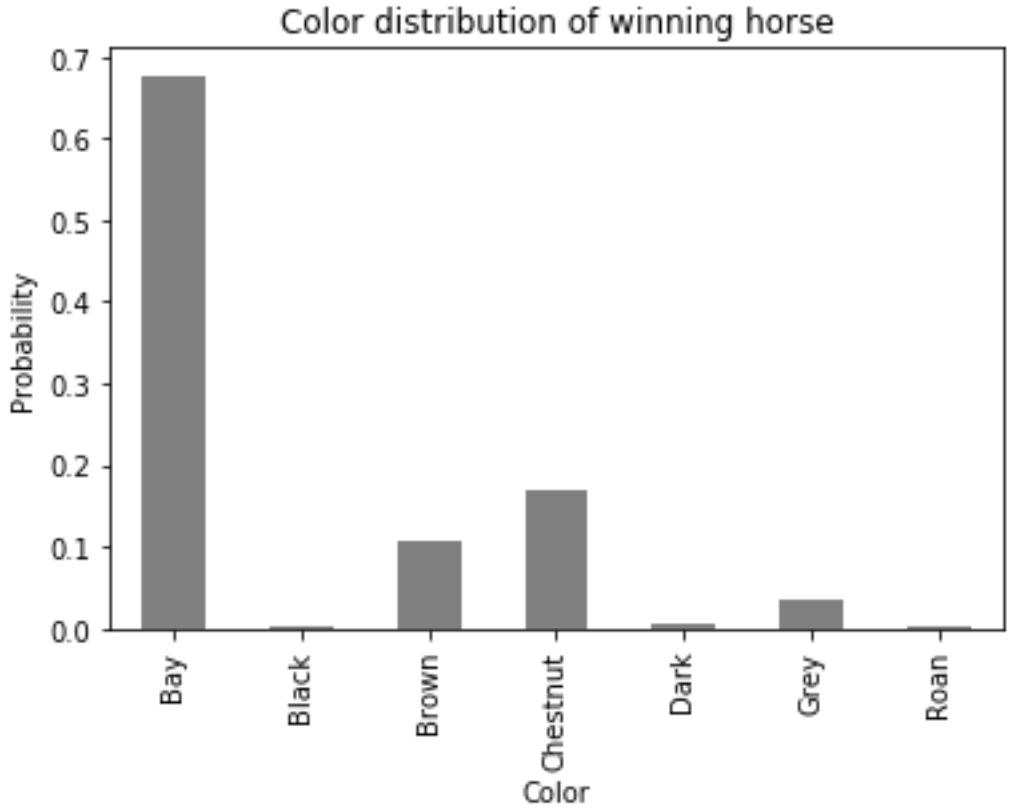


Figure 6. Color distribution of winning horse

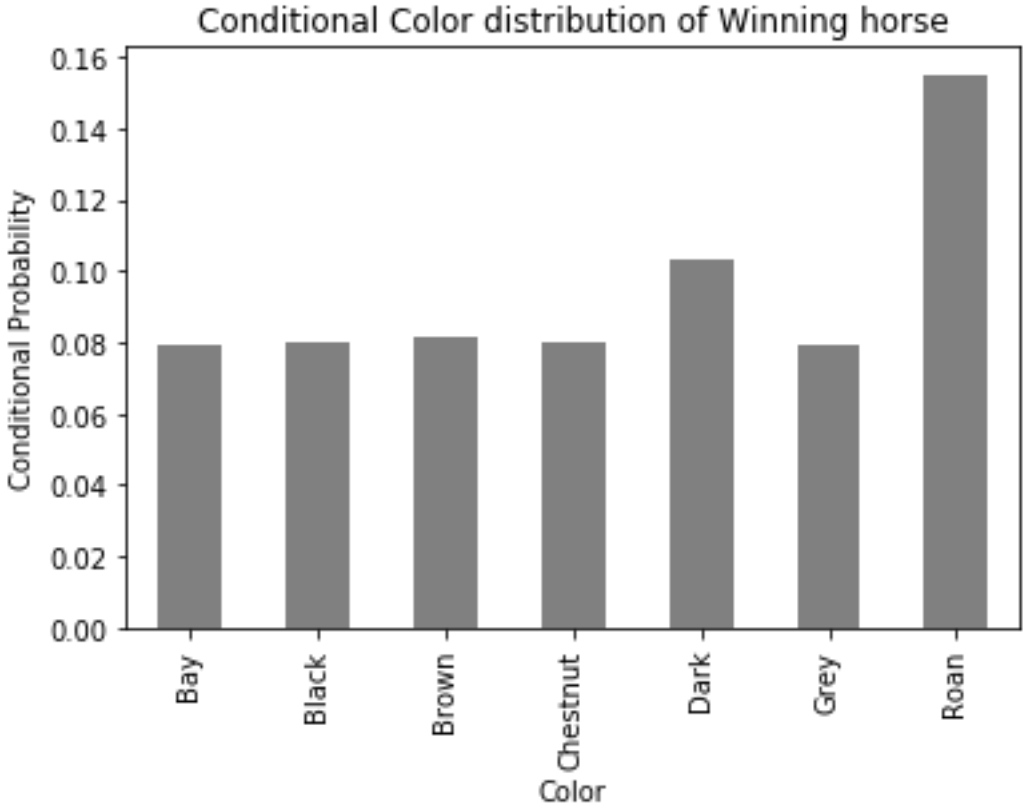


Figure 7. Conditional Color distribution of winning horse

5.3.4 Sex

The sex of the horse is mainly classified into Gelding, Colt, or Filly. Over 90% of the runners in Hong Kong are geldings [27]. The different hormones levels of different sex may lead to different performance [28]. To analyze whether sex affects winning performance of the horses, the sex distribution and conditional sex distribution of winning horse is shown on the next page. The analysis of sex distribution of winning horse reveals that sex is also important in determination of winners and should be included for the model input. In general, it can be inferred that male horses (Colt, Gelding, Horse, Rig) has a higher winning probability than female horses (Filly, Mare).

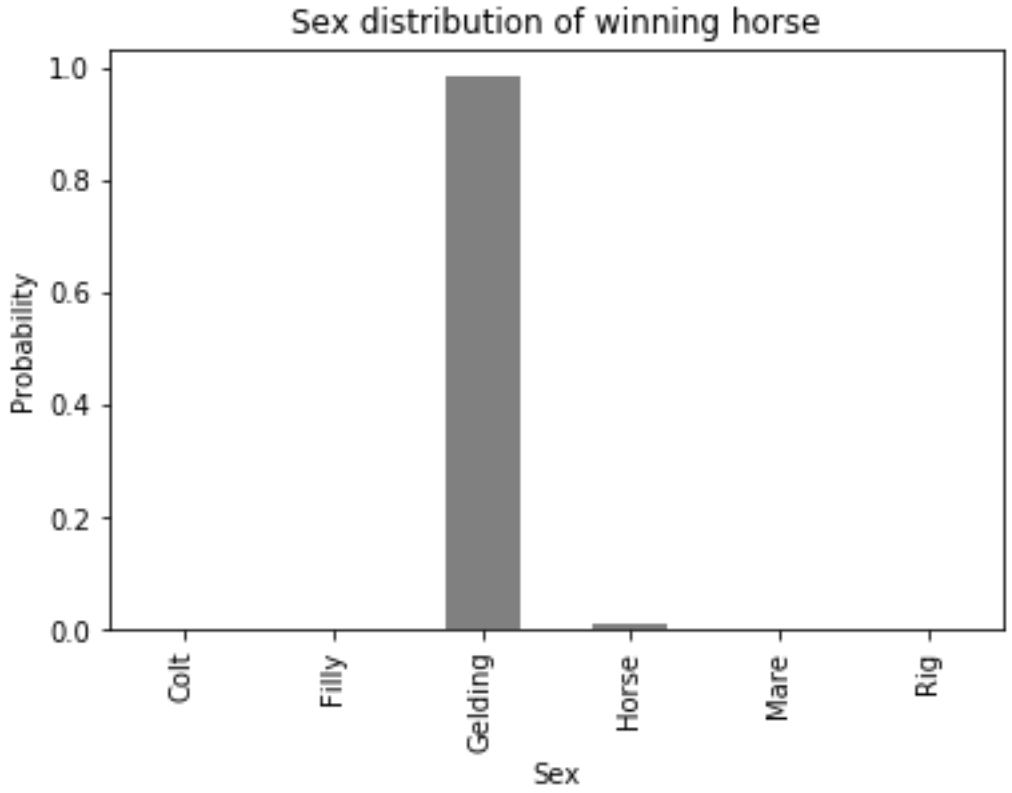


Figure 8. Sex distribution of winning horse

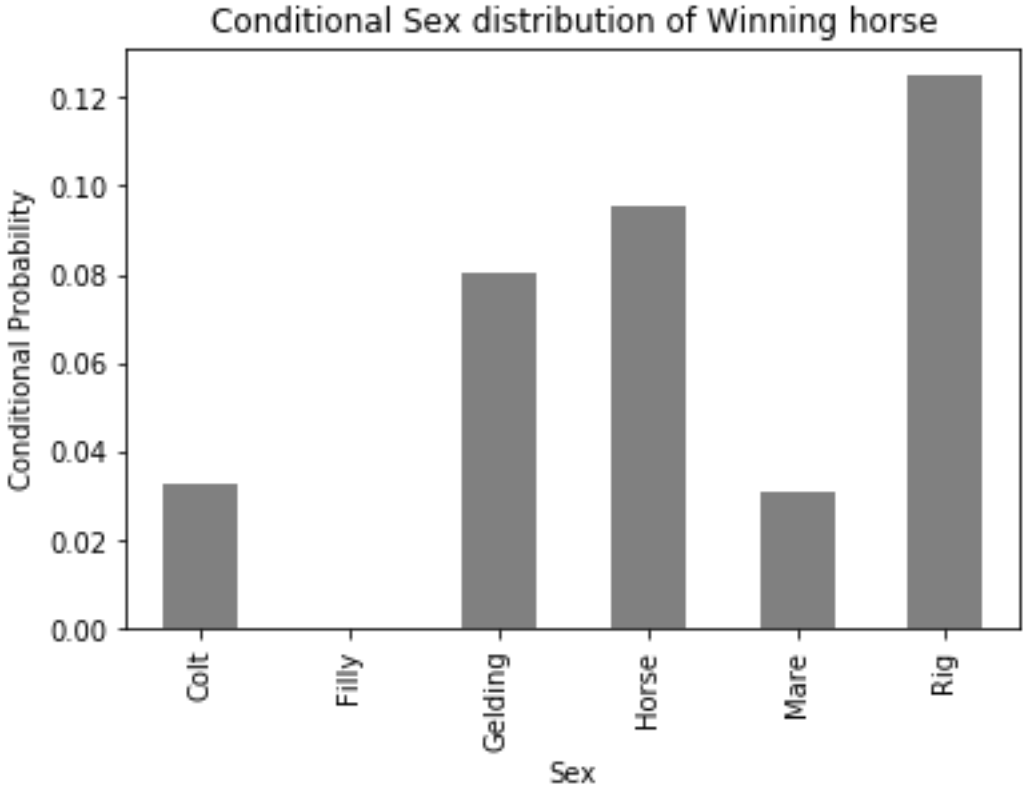


Figure 9. Conditional Sex distribution of winning horse

5.3.5 Draw

In general, horses starting with an inside draw (smaller draw number) have a competitive advantage, since an inside rail has a shorter distance at turns [29]. However, the distance and the running style of the horse may also impact the influence of draw number. For example, Shatin Turf 1000M Straight has no turns and there is no advantage for having an inside draw. In addition, as there is less damage to the track on the outskirts of the track, horses that start from an outside draw (larger draw number) have a competitive advantage.

To verify whether the above principle is correct, we plot the draw distribution and the conditional draw distribution of winning horse.

The figures indicate that horses with smaller draw number are more likely to win, which supports the general principle of [27]. Therefore, it can be concluded that draw is indeed an important feature and should be included for the model input.

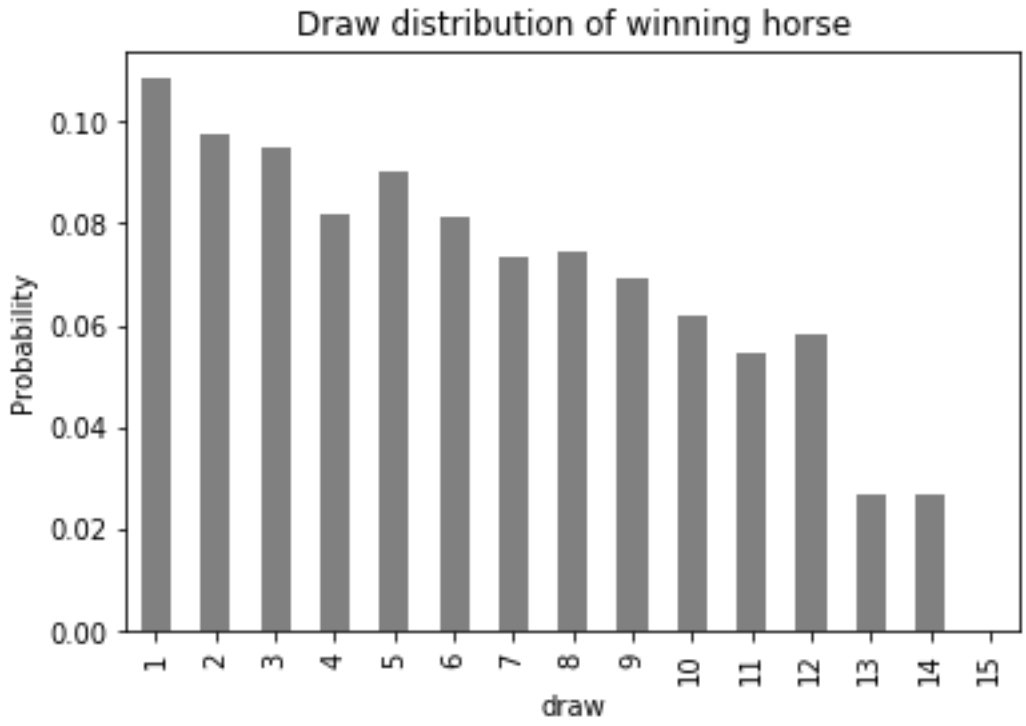


Figure 10. Draw distribution of winning horse

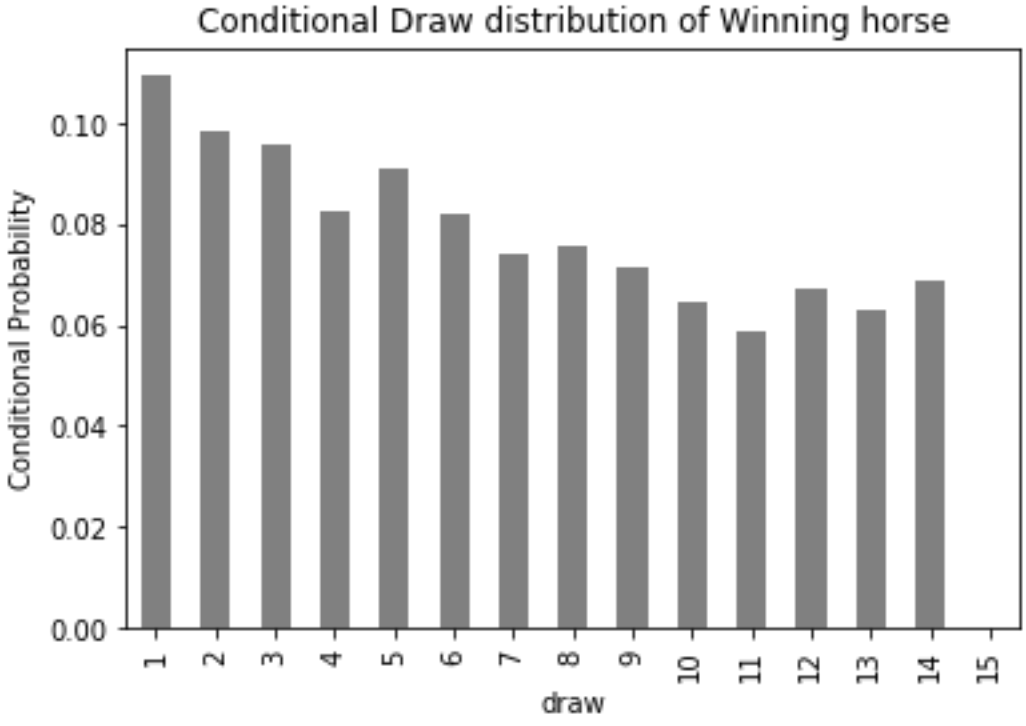


Figure 11. Conditional Draw distribution of winning horse

5.3.6 Old place

Apart from the intrinsic characteristic of the horse, the past performance of the horse is also important. Intuitively, a horse with a track record of all first places is more likely to win then a horse with a track record of all last places.

To verify whether our intuition is correct, we plot the old place distribution of winning horse and the conditional old place distribution of winning horse below. Here, –1 indicates that there is no past record for the horse.

The data shown has clearly points out that winners will remain winners, and losers will remain losers. Therefore, the old place of the horse is also an important feature for prediction of horse place and should be included for model input.

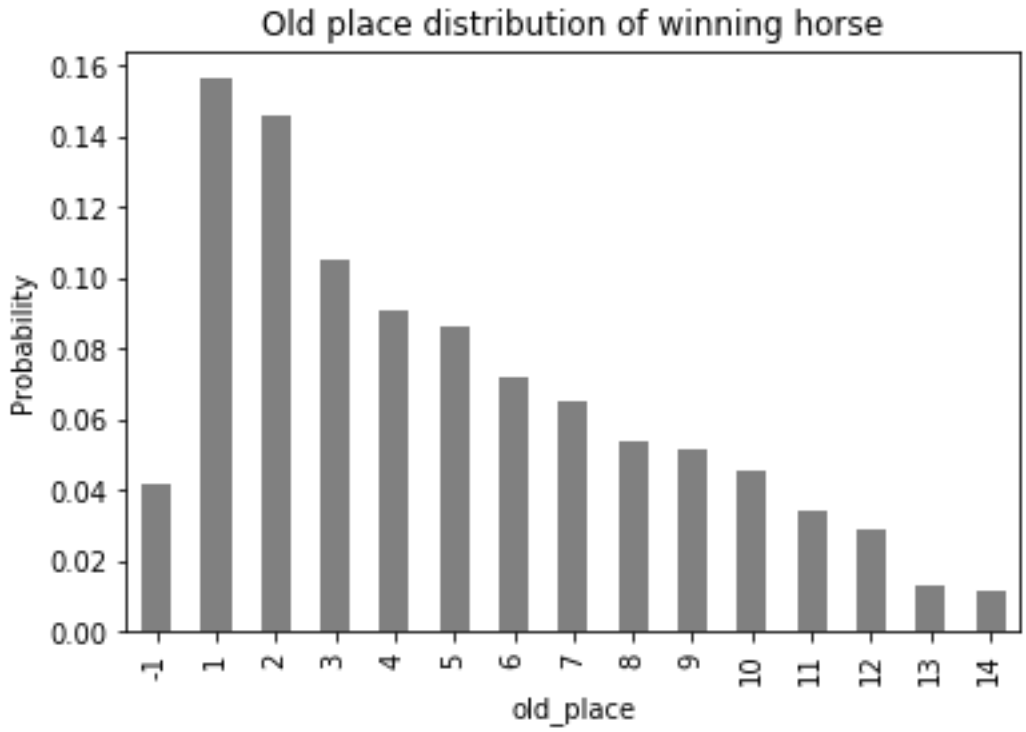


Figure 12. Old place distribution of winning horse

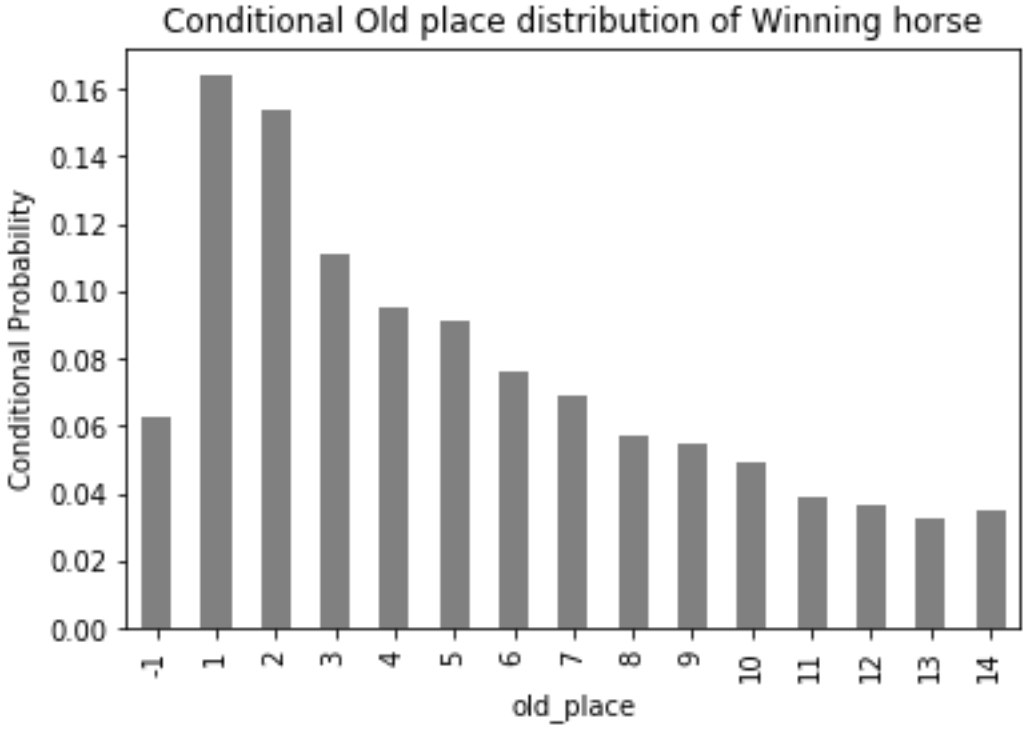


Figure 13. Conditional Old place distribution of winning horse

5.4 Feature Selection

5.4.1 Excluded Features

Some of the features in the data are not used for the model input. Features like raceyear are excluded as they are artificial numbering system. On the other hand, features like horseid, sire, dam are excluded as our past result from [9] shown below in indicates that they can only slightly improve accuracy but at the same time decrease the net gain. In the table below, Features Set A contains no identity features, B contains only human identity features, and C contains both human and horse identity features.

Feature Set	A	A+Odds	B	B+Odds	C	C+Odds
Accuracy	0.1840	0.2576	0.1798	0.2592	0.1830	0.2634
Net Gain	–184.68	–184.5	–177.45	–164.65	–220.29	–188.06
Return/Bet	–0.2165	–0.2163	–0.2080	–0.2009	–0.2583	–0.2205

Table 10. Past results of different input features from [9].

Therefore, the complete list of excluded features is shown below.

Feature	Description	Reason
raceyear	Year of the race	Artificial numbering system
racemonth	Month of the race	Replaced by weather features
raceday	Day of the race	Replaced by weather features
raceid	Unique id of the race	Artificial numbering system
raceno	Race number in a race day	Artificial numbering system
horseno	Number assigned by HKJC to horse	Artificial numbering system
horseid	Unique id of horse	Too many distinct values / cannot handle unseen horses
place	Place of horse in race	Information cannot be gained before the race
finishtime	Finishing time of horse	Same reason as place
sire	Father of horse	Same reason as horseid
dam	Mother of horse	Same reason as horseid
dam's sire	Maternal grandfather of horse	Same reason as horseid

Table 11. Excluded features

5.4.2 Included Features

The included features used for model input are listed in the following table.

Feature	Description	Types	Values
location	Location of the race	Categorical	ST, HV
class	Class of the horses	Categorical	Class 1 to 5, Group 1 to 3
distance	Distance of the race	Categorical	1000, 1200, 1400, 1600, 1650, 1800, 2000, 2200, 2400
course	Track used for the race	Categorical	A, A+3, AWT, B, B+2, C, C+3
going	Soil measurement	Categorical	FIRM, GOOD TO FIRM, GOOD, GOOD TO YIELDING, YIELDING, YIELDING TO SOFT, FAST, SLOW, WET FAST, WET SLOW
jockeycode	Unique id of jockey	Categorical	169 distinct values
trainercode	Unique id of trainer	Categorical	147 distinct values
draw	Draw of the horse in race	Categorical	1 to 14
actualweight	Weight added to horse	Real value	–
horseweight	Weight of horse itself	Real value	–
winodds	Bet return on win	Real value	1 to 99
origin	Place of origin	Categorical	
age	Age of horse	Real value	3 to 10
color	Color of horse	Categorical	Bay, Black, Brown, Chestnut, Dark, Grey, Roan
sex	Sex of horse	Categorical	Colt, Filly, Gelding, Horse, Mare, Rig
temperature	Air temperature	Real value	–
weather	Description of weather	Categorical	Bay, Black, Brown, Chestnut, Dark, Grey, Roan
wind _speed	Wind speed in km/h	Real value	–
wind_direction	Wind direction	Categorical	–
humidity	Humidity	Real value	0 to 100
moon	Moon phase	Real value	0 to 29.5305882
dn	Day or Night	Categorical	D, N
old_place	Place of horse in last race	Categorical	1 to 14
weightdiff	Difference in weight from previous race	Real value	–

Table 12. Included features

5.4.3 Investigated Feature

Compared to our past work [9], we have added a set of weather features for model input. The features under investigate are listed in the table below.

Feature	Description	Reason
temperature	Air temperature	Investigate effect of weather
weather	Description of weather	Investigate effect of weather
wind _speed	Wind speed in km/h	Investigate effect of weather
wind_direction	Wind direction	Investigate effect of weather
humidity	Humidity	Investigate effect of weather
moon	Moon phase	Investigate effect of weather
dn	Day or Night	Investigate effect of weather
winodds	Bet return on win	Investigate influence of public intelligence

Table 13. Investigated features

To compare the effect of adding such features, we create three sets of features for model input, all features, without “winodds” feature, and without weather features. The list of used features in each features set is shown in the following table. The total number of features in asset and the resultant input dimension per horse is also included.

Feature\Feature Set	All Features	Without “winodds"	Without Weather
location	X	X	X
class	X	X	X
distance	X	X	X
course	X	X	X
going	X	X	X
jockeycode	X	X	X
trainercode	X	X	X
draw	X	X	X
actualweight	X	X	X
horseweight	X	X	X
winodds	X		X
origin	X	X	X
age	X	X	X
color	X	X	X
sex	X	X	X
temperature	X	X	
weather	X	X	
wind _speed	X	X	
wind_direction	X	X	
humidity	X	X	
moon	X	X	
dn	X	X	
old_place	X	X	X
weightdiff	X	X	X
Total features	24	23	17
Input Dimension	455	454	391

Table 14. Features used in each feature set

5.5 Data Preprocessing

5.5.1 Real Value Data

We apply normalization on real value data to make training less sensitive to the scale of individual features. We use the z–score normalization to make the data have zero mean and unit variance. To prevent information leakage, we use the mean and variance of the training data for normalization. The data is then normalized according to the following equation:

$$\hat{X} = \frac{X - mean(X)}{std(X)}$$

5.5.2 Categorical Data

We use one hot encoding to represent categorical data. This approach, while creating a high dimension and memory intensive, represents categorical data in an unbiased way so that every class is equally separated and unrelated. This approach is also the most straight forward way to represent categorical data. For example, suppose we have data of different categories as follows:

Item	Category
1	Apple
2	Orange
3	Banana

After one hot encoding, the data will be transformed into the following:

Item	Is Apple	Is Orange	Is Banana
1	1	0	0
2	0	1	0
3	0	0	1

Note that the dimension of the data is increased by the number of categories.

After this step, the dimension of the data is increased by over 20 times from 21 to 455. One approach to overcome the high dimensionality and large memory consumption is to train an embedding network for each of column of the data set. However, this requires careful selection of embedding dimension and complicated network design. Since our dataset are still well within the size of available memory, it is not deemed as necessary to use embedding networks.

Chapter 6 Design

In this chapter, we describe the design of our model and explain the design decisions made. In addition, we highlight the key design differences of our model with related works.

6.1 Design Goals

The design goals of our model are to accurately predict the winning horse and to generate profit with the “win” bet provided by the Hong Kong Jockey Club. All subsequent design choices are evaluated by 1. Accuracy of the prediction of the winning horse and 2. Simulated testing return with the “win” bet provided by the Hong Kong Jockey Club.

6.2 Race Representation

In this section we describe the different representations of horse races. In previous studies [7] [8], regression on finishing time and binary classification on win/lose are mainly studied, while our previous work [9] focused on multi–class classification of place to model horse performance.

6.2.1 Single Horse Representations

1. Finishing time regression – Regression on finishing time is a simple yet effective way to interpret horse racing results. In this approach, finishing time of each individual horse are predicted and the horses are ranked based on the predicted time.
2. Win/lose binary classification – Binary classification on win/lose is another straightforward way to predict whether the horse is going to win. However, binary labeling the data of win/lose will result in highly uneven distributed labels with less than 10% of positive data and more than 90% of negative data.
3. Place prediction – Directly predicting the place of the horses is more complicated method but give even data to each class. Moreover, score of each place of a horse can be interpreted as the probability of the horse getting each place, which facilitates building a probabilistic model. Although this may result in duplicated place within the same race, the score for each place can be used for ranking the horses in a race. However, in races with less than the maximum number (14) of horses, this model is not coherent with intuition and gives probability for impossible places.

6.2.2 Multiple Horses Representations

All of the above methods predict their result based on the information of a single horse, which is unable to capture the interaction of different horses in a real race. In addition, the small errors in each individual horse may accumulate to lead to a large error in prediction. In this project, instead of using single horse representation, our model predicts the race result based on information from all the horses. However, this requires different models for prediction of races with different number of horses because of the difference in input and output dimension caused by the different number of horses. This is amortized by most races having 12 or 14 horses, as shown below in Figure 14. Therefore, by building two separate models for races of 12 horses and 14 horses, we can handle over 75% of the races.

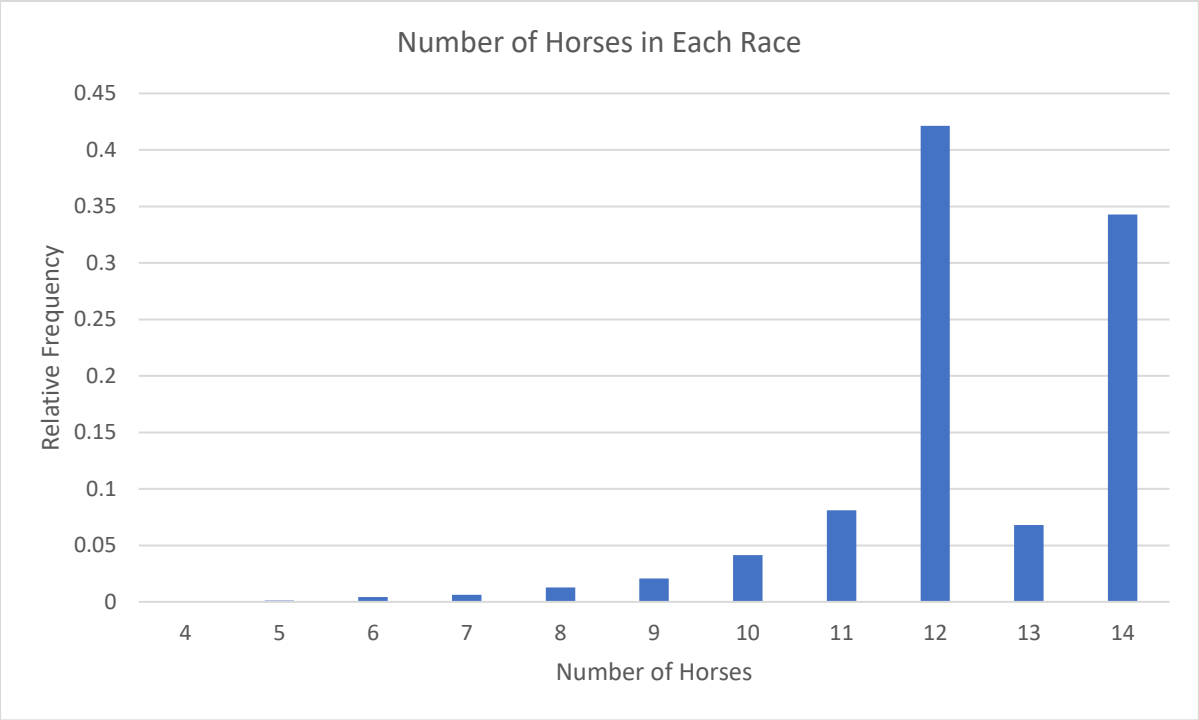


Figure 14. Number of horses in each race

For multiple horses, there are also different possible representations:

1. Multiple horse finishing time regression – Regression on finishing time is a simple yet effective way to interpret horse racing results. In this approach, finishing time of each individual horse are predicted and the horses are ranked based on the predicted time. However, this leads to a difficult choice of activation function for the output layer and may requires an additional transformation from the output of the model to the final output for training.

2. Winning horse classification – Binary classification on win/lose is a straightforward way to predict whether the horse is going to win. In addition, in the multiple horse case, binary labeling the data with win/lose will not result in uneven data labels, since in each race, there is always a horse which wins.
3. Multiple horses place prediction – Directly predicting the place of the horses is more complicated method that predict more information about the relative rankings of the horses apart from the winning horse. However, it requires a two–dimensional output layer for representing place probability for each of the horses.

In this project, we choose to use winning horse classification because it is the simplest representation that avoids the difficult choice of activation functions and the two–dimensional output layer while still being able to predict the winning horse. In addition, while finishing time regression and place prediction can yield additional information about the rankings of individual horses, we are only concerned with the prediction of the winning horse and betting on the "win" bet of Hong Kong Jockey Club in this project.

6.3 Network Design

6.3.1 Distribution Selection for Neural Network Parameters

Han et al. [30] [31] have studied the weight distribution of three state of the art deep neural networks, LeNet [32], AlexNet [33] and VGGNet [34], shown below in Figure 15.

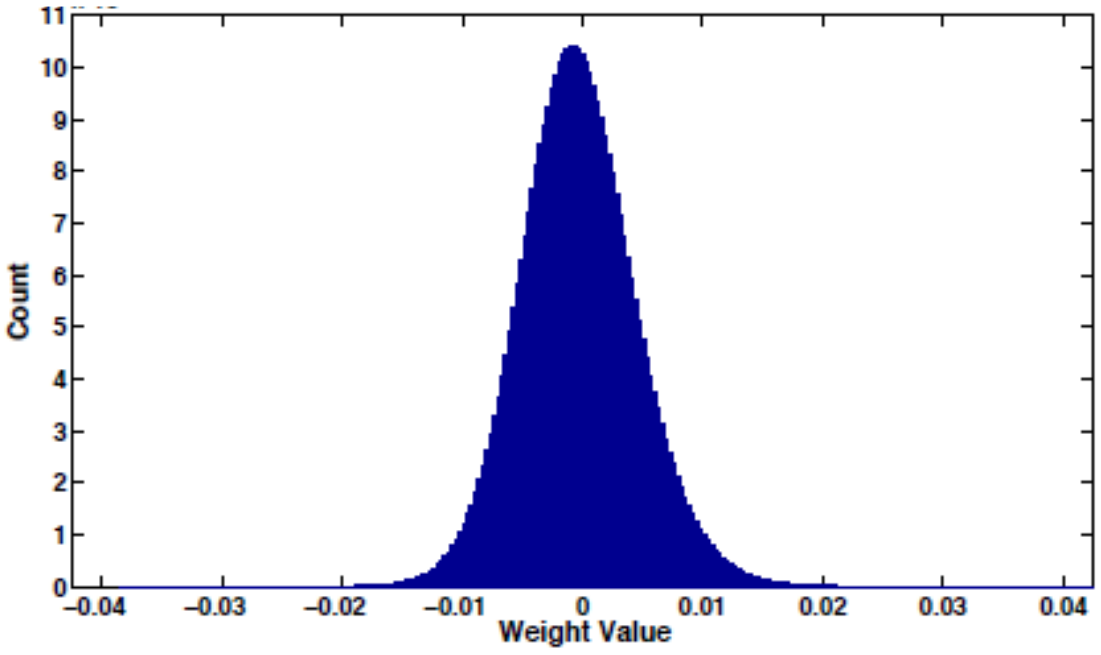


Figure 15. Distribution of weights in different deep neural networks

The weight distribution in these neural networks has been found to resemble narrow normal distributions with mean very close to zero. Therefore, in our model, both the prior distributions and variational distributions are set to be normal distributions.

On the other hand, despite narrow distribution of parameters found by Han et al. [30] [31] with standard deviation less than 0.01, we set our prior distribution to be fairly wide with standard deviation of 1, Normal(0,1), to accommodate a larger range of possible values.

6.3.2 Number of Layers and Neurons in Network

In our previous work [9], we have determined that neural network with 4 layers of which 3 of them hidden works best. Therefore, in this work, we retain the same number of layers in the neural network. Instead, we test different number of neurons each layer ranging from 16 to 256 and select the best number of neurons. Figure 16 below illustrates our 4–layer network structure with 16 neurons per layer. The dimension of the input layer in the figure has been reduced to 20 for a better presentation. The output of this neural network will then be used as rank probabilities during training. The complete flow is shown in Figure 17.

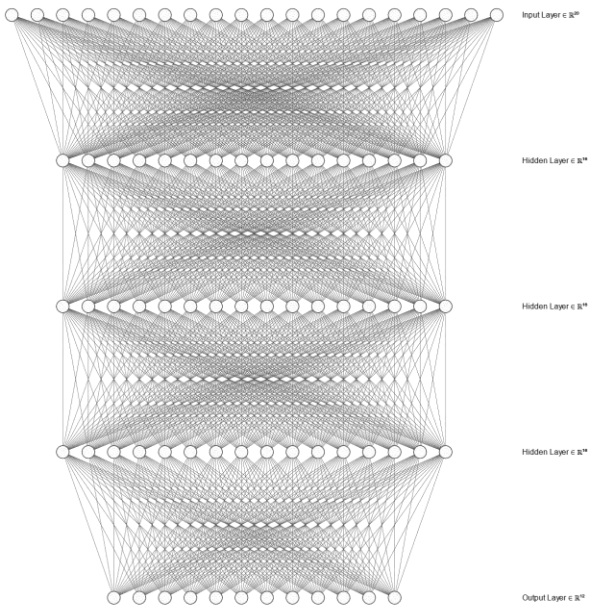


Figure 16. Our 4–layer neural network with 16 neurons per layer

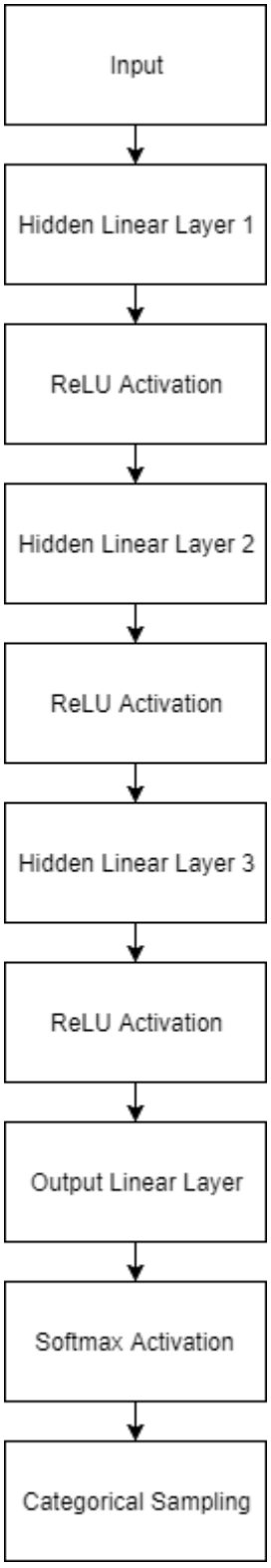


Figure 17. Flow of our neural network with final categorical sampling

Chapter 7 Implementation

We have implemented our model on Python 3.7 with PyTorch 1.0.1 and Pyro 0.3.1. In this section, we describe the process of translating our model to implementation on Pyro.

7.1 Pyro

Pyro is a probabilistic programming language build on Python as a platform for developing advanced probabilistic models [22]. It leverages PyTorch on the backend to support neural networks, backpropagation, and automatic differentiation. As a probabilistic programming language, it abstracts probabilistic sampling and inference with simple primitives. For example, only one statement is needed in Pyro to sample a Normal(0,1) distribution,

```
x = pyro.distributions.Normal(0.0, 1.0).sample()
```

Given the model `model` and variational distribution `guide`, (stochastic) variational inference can be done with a few lines as follows:

```
svi = SVI(model, guide, optimizer, loss=Trace_ELBO())
for i in range(num_epochs):
    svi.step(data)
```

Similarly, Markov Chain Monte Carlo sampling can be done as simple as:

```
nuts_kernel = pyro.infer.mcmc.NUTS(model, adapt_step_size=True)
posterior = pyro.infer.mcmc.MCMC(nuts_kernel, num_samples=num_epochs).run(data)
```

Running either would yield the approximate posterior distribution $p(z|x)$, stored in `guide` or `posterior`, depending on whether variational inference or Markov Chain Monte Carlo is used.

7.2 Bayesian Neural Network in Pyro

The following code implements our Bayesian neural network as described in Section 6.3 :

```
class Net(Module):
    def __init__(self, num_feature, num_hidden, num_rank):
        super(Net, self).__init__()
        self.hidden1 = Linear(num_feature, num_hidden)   # hidden layer 1
        self.hidden2 = Linear(num_hidden, num_hidden)   # hidden layer 2
        self.hidden3 = Linear(num_hidden, num_hidden)   # hidden layer 3
        self.predict = Linear(num_hidden, num_rank)   # output layer
        self.nonlinear = ReLU()   # ReLU activation for intermediate layers
        self.softmax = Softmax(1) )   # Softmax activation for final layer

    def forward(self, x):
        hid1 = self.nonlinear(self.hidden1(x))
        hid2 = self.nonlinear(self.hidden2(hid1))
        hid3 = self.nonlinear(self.hidden3(hid2))
        pred = self.softmax(self.predict(hid3))
        return pred

def model(data):
    # Create unit normal priors over the parameters
    loc = pyro.zeros(num_hidden, num_feature)
    scale = pyro.ones(num_hidden, num_feature)
    bias_loc = pyro.zeros(num_hidden)
    bias_scale = pyro.ones(num_hidden)
    loc2 = pyro.zeros(num_hidden, num_hidden)
    scale2 = pyro.ones(num_hidden, num_hidden)
    bias_loc2 = pyro.zeros(num_hidden)
    bias_scale2 = pyro.ones(num_hidden)
    loc3 = pyro.zeros(num_hidden, num_hidden)
    scale3 = pyro.ones(num_hidden, num_hidden)
    bias_loc3 = pyro.zeros(num_hidden)
    bias_scale3 = pyro.ones(num_hidden)
    loc_out = pyro.zeros(num_rank, num_hidden)
    scale_out = pyro.ones(num_rank, num_hidden)
    bias_loc_out = pyro.zeros(num_rank)
    bias_scale_out = pyro.ones(num_rank)

    w_prior = Normal(loc, scale).independent(2)
    b_prior = Normal(bias_loc, bias_scale).independent(1)
    w_prior2 = Normal(loc2, scale2).independent(2)
    b_prior2 = Normal(bias_loc2, bias_scale2).independent(1)
    w_prior3 = Normal(loc3, scale3).independent(2)
    b_prior3 = Normal(bias_loc3, bias_scale3).independent(1)
    w_prior_out = Normal(loc_out, scale_out).independent(2)
    b_prior_out = Normal(bias_loc_out, bias_scale_out).independent(1)
    priors = {'hidden1.weight': w_prior, 'hidden1.bias': b_prior, 'hidden2.weight':
w_prior2, 'hidden2.bias': b_prior2, 'hidden3.weight': w_prior3, 'hidden3.bias':
b_prior3, 'predict.weight': w_prior_out, 'predict.bias': b_prior_out}

    lifted_module = pyro.random_module("module", Net, priors)
    lifted_reg_model = lifted_module()
    x_data = data[:, :-1]
    y_data = data[:, -1]
    with pyro.plate("map", len(data)):
        prediction_mean = lifted_reg_model(x_data)
        pyro.sample("obs", Categorical(prediction_mean), obs=y_data)
```

Figure 18. Implementation of Bayesian neural network model in Pyro

```
def guide(self, data):
    # define our variational parameters
    w_loc = data.new_tensor(pyro.randn(num_hidden, num_feature))
    w_log_sig = data.new_tensor(pyro.zeros(num_hidden, num_feature))
    b_loc = data.new_tensor(pyro.randn(num_hidden))
    b_log_sig = data.new_tensor(pyro.zeros(num_hidden))
    w_loc2 = data.new_tensor(pyro.randn(num_hidden, num_hidden))
    w_log_sig2 = data.new_tensor(pyro.zeros(num_hidden, num_hidden))
    b_loc2 = data.new_tensor(pyro.randn(num_hidden))
    b_log_sig2 = data.new_tensor(pyro.zeros(num_hidden))
    w_loc3 = data.new_tensor(pyro.randn(num_hidden, num_hidden))
    w_log_sig3 = data.new_tensor(pyro.zeros(num_hidden, num_hidden))
    b_loc3 = data.new_tensor(pyro.randn(num_hidden))
    b_log_sig3 = data.new_tensor(pyro.zeros(num_hidden))
    w_loc_out= data.new_tensor(pyro.randn(num_rank, num_hidden))
    w_log_sig_out= data.new_tensor(pyro.zeros(num_rank, num_hidden))
    b_loc_out= data.new_tensor(pyro.randn(num_rank))
    b_log_sig_out= data.new_tensor(pyro.zeros(num_rank))

    # register learnable params in the param store
    mw_param = pyro.param("guide_mean_weight", w_loc)
    sw_param = Softplus(pyro.param("guide_log_scale_weight", w_log_sig))
    mb_param = pyro.param("guide_mean_bias", b_loc)
    sb_param = Softplus(pyro.param("guide_log_scale_bias", b_log_sig))
    mw_param2 = pyro.param("guide_mean_weight2", w_loc2)
    sw_param2 = Softplus(pyro.param("guide_log_scale_weight2", w_log_sig2))
    mb_param2 = pyro.param("guide_mean_bias2", b_loc2)
    sb_param2 = Softplus(pyro.param("guide_log_scale_bias2", b_log_sig2))
    mw_param3 = pyro.param("guide_mean_weight3", w_loc3)
    sw_param3 = Softplus(pyro.param("guide_log_scale_weight3", w_log_sig3))
    mb_param3 = pyro.param("guide_mean_bias3", b_loc3)
    sb_param3 = Softplus(pyro.param("guide_log_scale_bias3", b_log_sig3))
    mw_param_out = pyro.param("guide_mean_weight_out", w_loc_out)
    sw_param_out = Softplus(pyro.param("guide_log_scale_weight_out", w_log_sig_out))
    mb_param_out = pyro.param("guide_mean_bias_out", b_loc_out)
    sb_param_out = Softplus(pyro.param("guide_log_scale_bias_out", b_log_sig_out))

    # guide distributions for w and b
    w_dist = Normal(mw_param, sw_param).independent(2)
    b_dist = Normal(mb_param, sb_param).independent(1)
    w_dist2 = Normal(mw_param2, sw_param2).independent(2)
    b_dist2 = Normal(mb_param2, sb_param2).independent(1)
    w_dist3 = Normal(mw_param3, sw_param3).independent(2)
    b_dist3 = Normal(mb_param3, sb_param3).independent(1)
    w_dist_out = Normal(mw_param_out, sw_param_out).independent(2)
    b_dist_out = Normal(mb_param_out, sb_param_out).independent(1)
    dists = {'hidden1.weight': w_dist, 'hidden1.bias': b_dist, 'hidden2.weight':
w_dist2, 'hidden2.bias': b_dist2, 'hidden3.weight': w_dist3, 'hidden3.bias': b_dist3,
'predict.weight': w_dist_out, 'predict.bias': b_dist_out}

    lifted_module = pyro.random_module("module", Net, dists)
    return lifted_module()
```

Figure 19. Implementation of variational distribution for Bayesian neural network in Pyro

7.3 Data Augmentation

To create more data points from available data, we employ data augmentation to transform data entries into additional entries. Data augmentation is often used in machine learning to prevent overfitting as a result of small training dataset, especially for tasks where manual labeling is expensive, such as image recognition [32] [33] [34]. However, for horse racing, data augmentation is a challenging task. Unlike image recognition where a human can easily distinguish between different labels given the input image, in horse racing the winning horse cannot be determined by a human given only the input data. In addition, for image recognition, noise filter placed over the input data does not drastically changes the label, but for horse racing, slight differences in the race condition can lead to different outcome. Therefore, we do not modify the values of the data, but only crop and shuffle the ground truth data.

7.3.1 Data Cropping

The presence of 13-horse races and 14-horse races present an opportunity for creating additional data for training 12-horse model. We crop the top 12 horses from 13-horse and 14-horse races for use in training the 12-horse model. As a result, the number of training races for the 12-horse model doubles from 2292 12-horse races to 4605 races of 12-horse, 13-horse and 14-horse combined. For the 14-horse model, there are no additional data that can be created from cropping because there is no race with more than 14 horses, so the number of training data for 14-horse model is only 1940 races.

7.3.2 Data Shuffling

In our multiple horse representation, the input features of different horses are symmetric, and there is no information encoded in the relative positions between the features of different horses. For example, for an input vector of 12 horses as follows:

Horse 1	Horse 2	Horse 3	Horse 4	Horse 5	Horse 6	Horse 7	Horse 8	Horse 9	Horse10	Horse11	Horse12

There is no difference in information encoded if the input vector is flipped:

Horse12	Horse11	Horse10	Horse 9	Horse 8	Horse 7	Horse 6	Horse 5	Horse 4	Horse 3	Horse 2	Horse 1

This enables an additional dimension for data shuffling.

Therefore, during training, the training data is shuffled along two dimensions, the order of training data and the order of each horse's features, before running each step of variational inference or Markov Chain Monte Carlo sampling.

Chapter 8 Results

8.1 Setup

We have implemented our model on Python 3.7 with PyTorch 1.0.1 and Pyro 0.3.1. All experimented are run on CentOS Linux 7 with Nvidia Titan V. We have trained our models with 100,000 epochs of variation inference over the training dataset with Adam optimizer with an initial learning rate of 0.0001. The amount of epochs is selected so that even the model with the largest number of parameters are trained to convergence. Since the variational distribution are not able to be directly used as network parameters, we sample the most likely set of weights and biases from our model by setting them to the mean of the variational distribution. Note that this is different from our previous approach [16], where 100 different sets of neural weights and biases from the variational distribution were sampled and the average performance taken as the result. We demonstrate in later sections that this approach of using the most likely model reduces the variability and improves the prediction accuracy and net gain.

8.2 Results

We use the following criteria to evaluate the different models:

1. Accuracy: accuracy of predicting the winning horse in testing dataset
2. Net gain: overall net gain of win over the testing dataset and ratio of return over bet

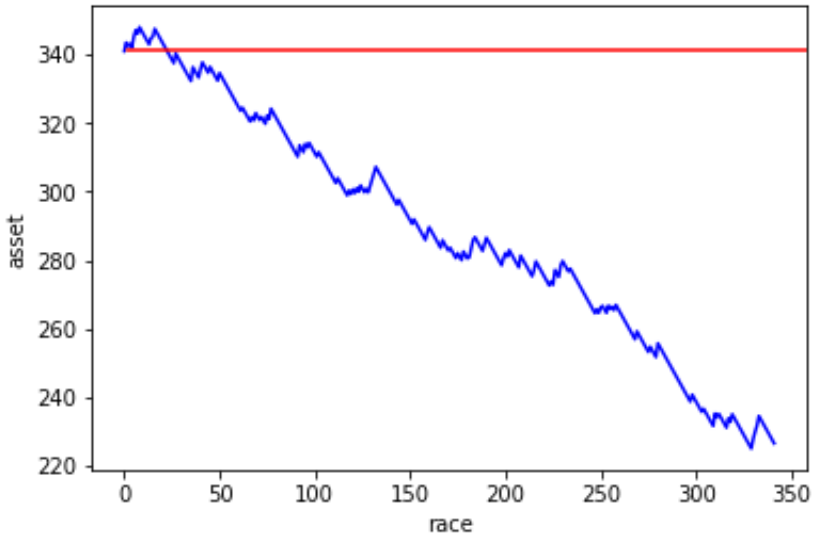
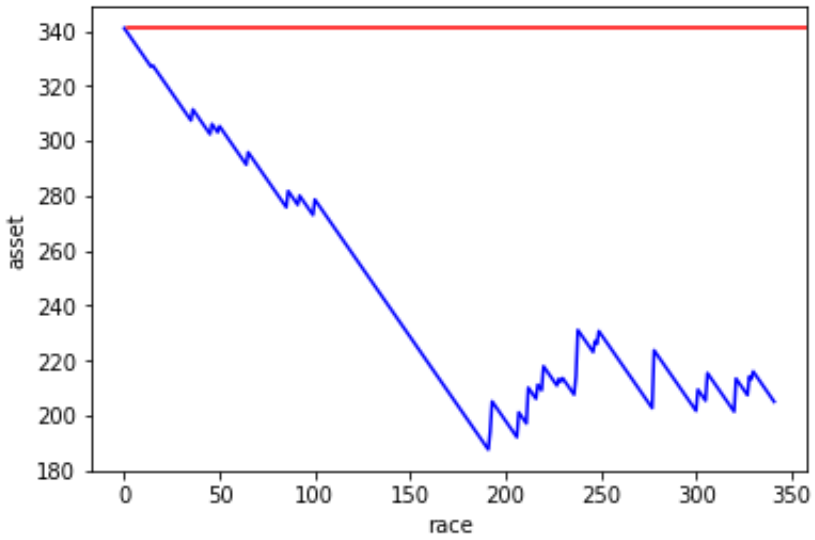
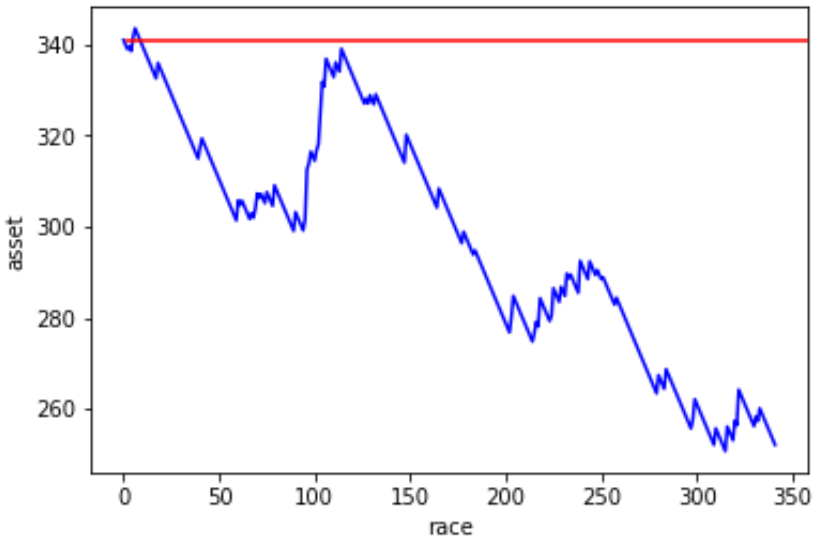
To generate the net gain, we bet a fixed amount of 1 for each race in the testing dataset on the predicted winning horse. This results in bet amount of 341 for 12-horse model and 203 for 14-horse model. The following tables show the performance of the model corresponding to the three set of features, All Features, Without “winodds” Feature, and Without Weather Features. The bet based on the public intelligence (lowest “winodds”) is also shown for comparison. To better illustrate the trend of our total asset during the testing process, we also included the betting curve showing the value of our total asset throughout the testing year.

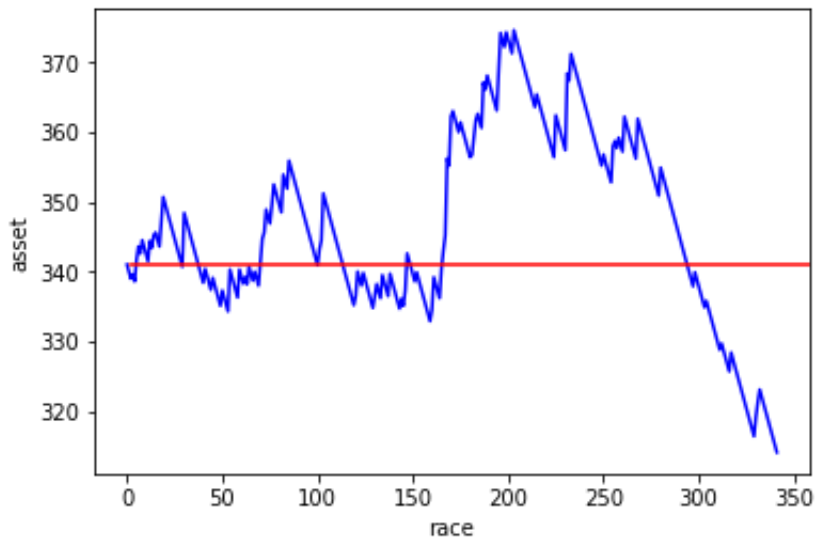
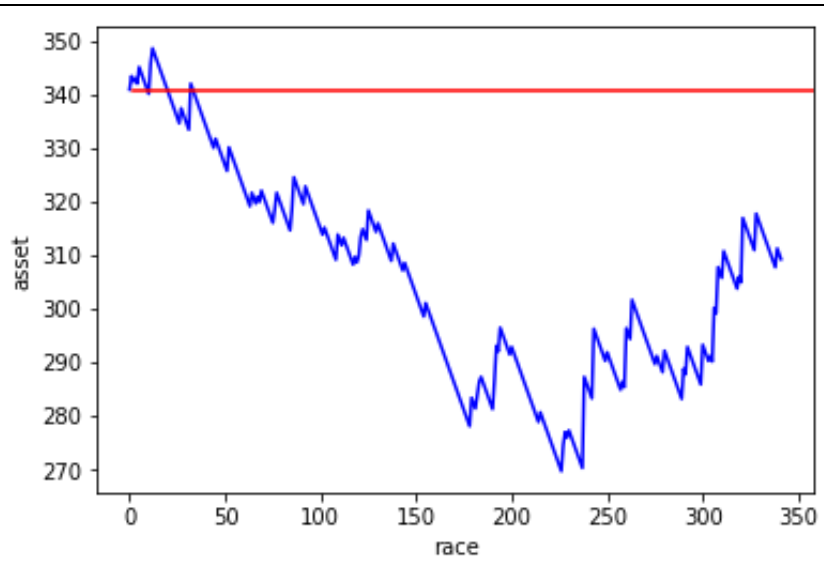
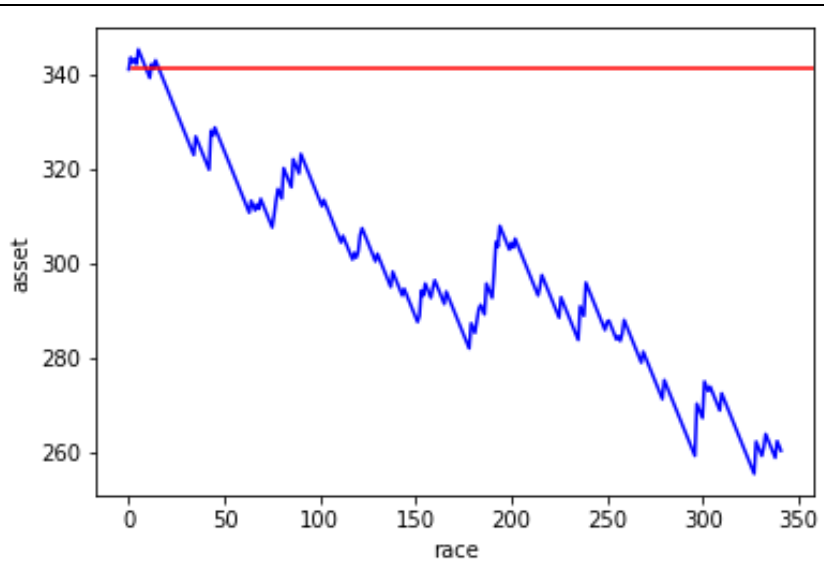
12-horse Model (Tested with 341 12-horse races)					
Feature Set	Neurons	Accuracy (%)	Net Return	Return/Bet (%)	Profit?
Public Intelligence	N/A	22.87	–114.5	–33.58	No
All Features	16	7.62	–136.0	–39.88	No
	32	17.01	–88.9	–26.07	No
	64	22.58	–26.9	–7.89	No
	128	17.60	–31.7	–9.30	No
	256	18.18	–80.6	–23.64	No
Without "winodds" Feature	16	8.80	–185.2	–54.31	No
	32	8.80	–185.2	–54.31	No
	64	20.82	–15.8	–4.63	No
	128	16.42	–67.6	–19.82	No
	256	17.60	–43.1	–12.64	No
Without Weather Features	16	9.68	–47.6	–13.96	No
	32	22.29	25.7	7.54	Yes
	64	19.35	–82.7	–24.25	No
	128	17.30	–73.5	–21.55	No
	256	17.30	–87.3	–25.60	No

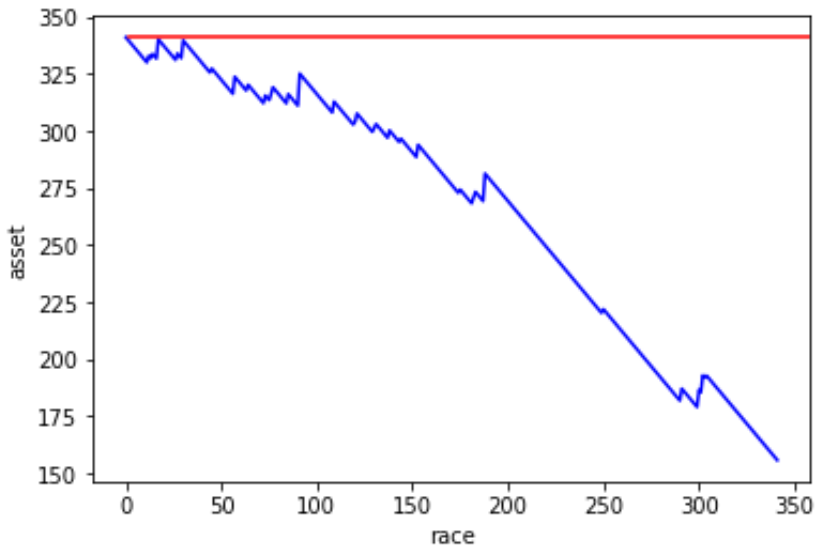
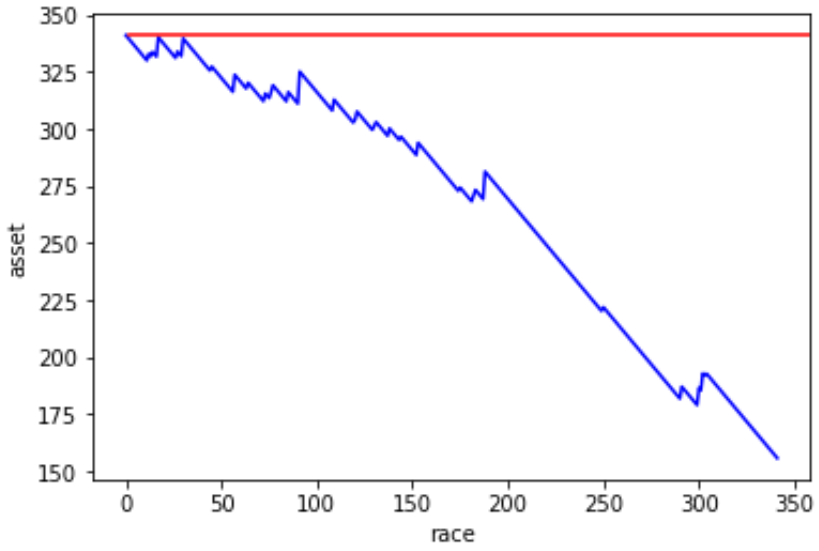
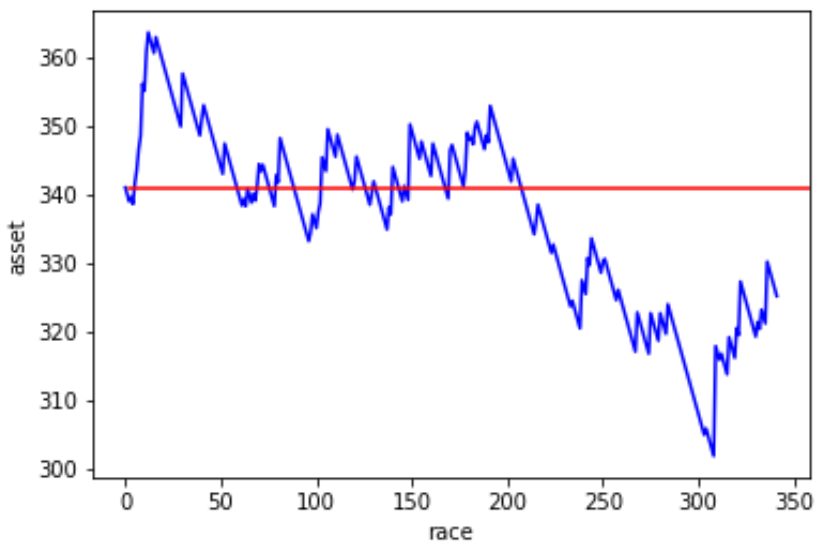
Table 15. Testing performance of 12-horse model with different features and number of neurons

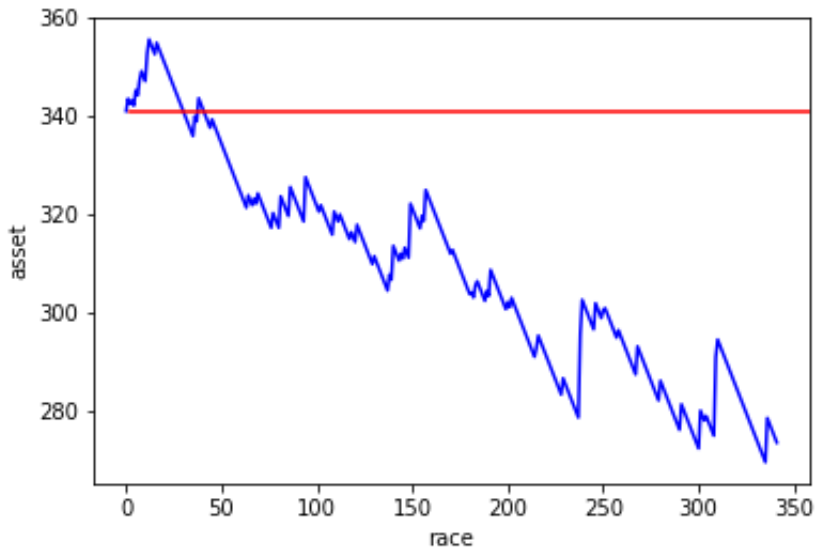
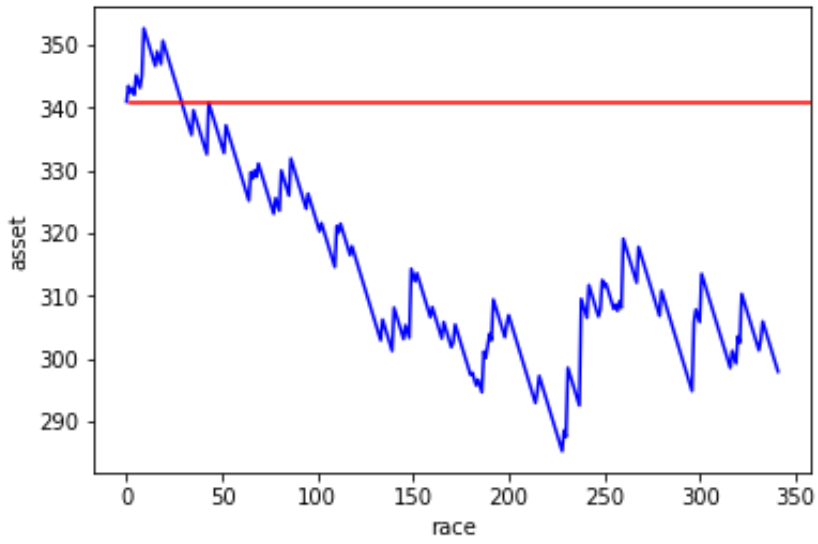
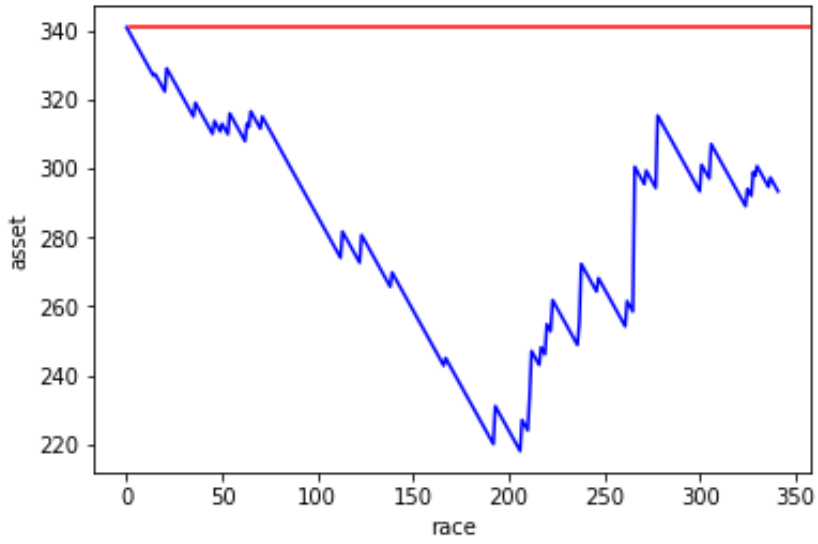
14-horse Model (Tested with 203 14-horse races)					
Feature Set	Neurons	Accuracy (%)	Net Return	Return/Bet (%)	Profit
Public Intelligence	N/A	27.59	–36.9	–18.18	No
All Features	16	5.91	–117.6	–57.93	No
	32	16.75	–41.6	–20.49	No
	64	14.29	–35.3	–17.39	No
	128	22.66	29.3	14.43	Yes
	256	17.24	–21.1	–10.39	No
Without "winodds" Feature	16	9.85	–117.7	–57.98	No
	32	15.76	–44.0	–21.67	No
	64	18.23	–27.5	–13.55	No
	128	23.15	15.4	7.59	Yes
	256	17.73	–38.6	–19.01	No
Without Weather Features	16	5.91	–117.6	–57.93	No
	32	20.20	–23.8	–11.72	No
	64	23.15	7.5	3.69	Yes
	128	15.76	–58.2	–28.67	No
	256	17.24	–49.7	–24.48	No

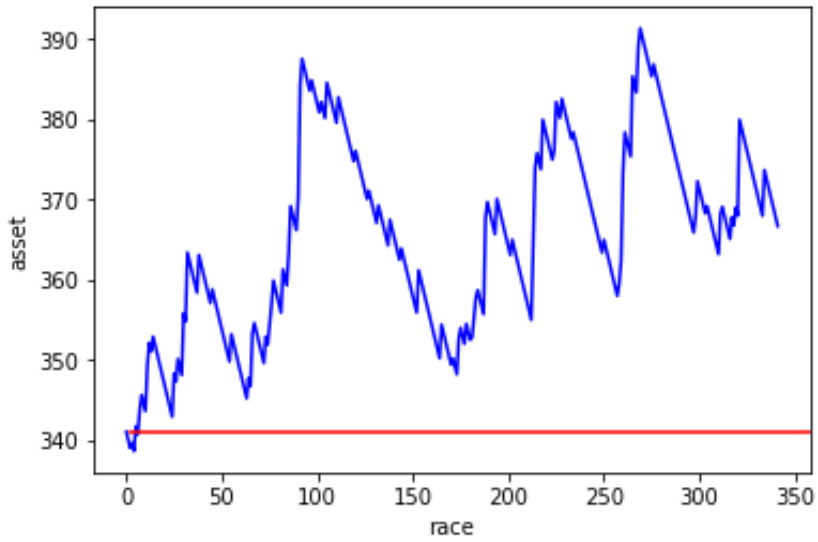
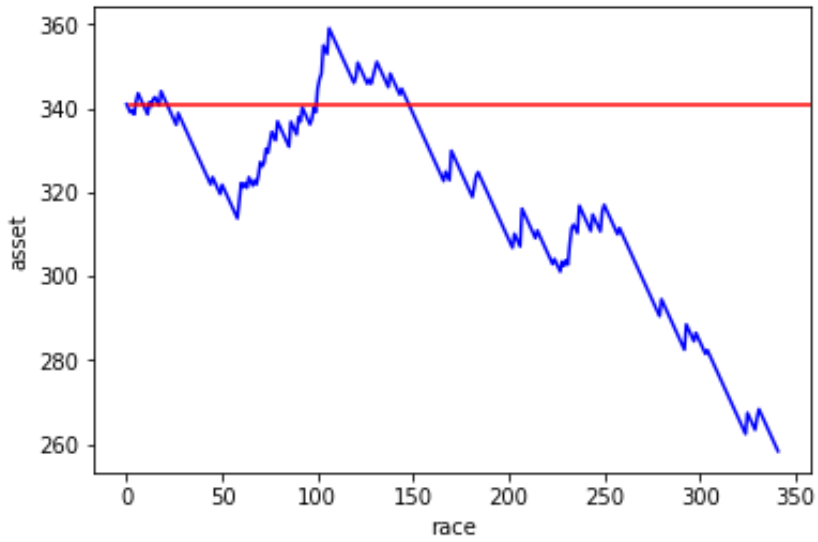
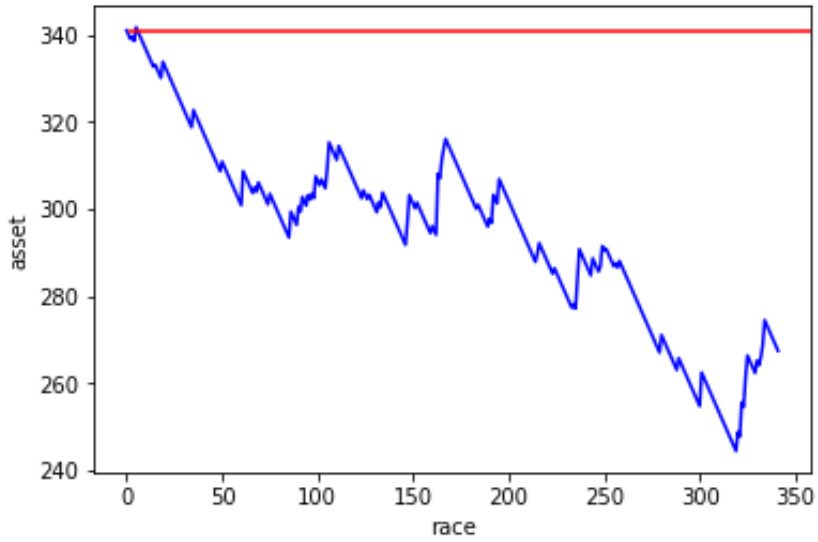
Table 16. Testing performance of 14-horse model with different features and number of neurons

<table>
<tr><td colspan="3">12-horse Model (Tested with 341 12-horse races)</td></tr>
<tr><td>Feature Set</td><td>Neurons</td><td>Betting Curve</td></tr>
<tr><td>Public Intelligence</td><td>N/A</td><td> 
</td></tr>
<tr><td rowspan="2">All Features</td><td>16</td><td> 
</td></tr>
<tr><td>32</td><td> 
</td></tr>
</table>

	64	
	128	
	256	

Without “winodds” Feature	16	
	32	
	64	

	128	
	256	
Without Weather Features	16	

	32	
	64	
	128	

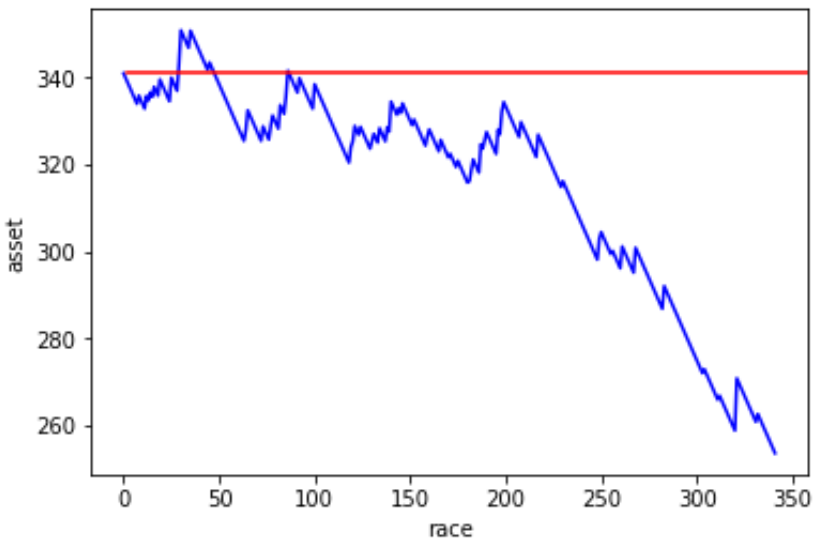
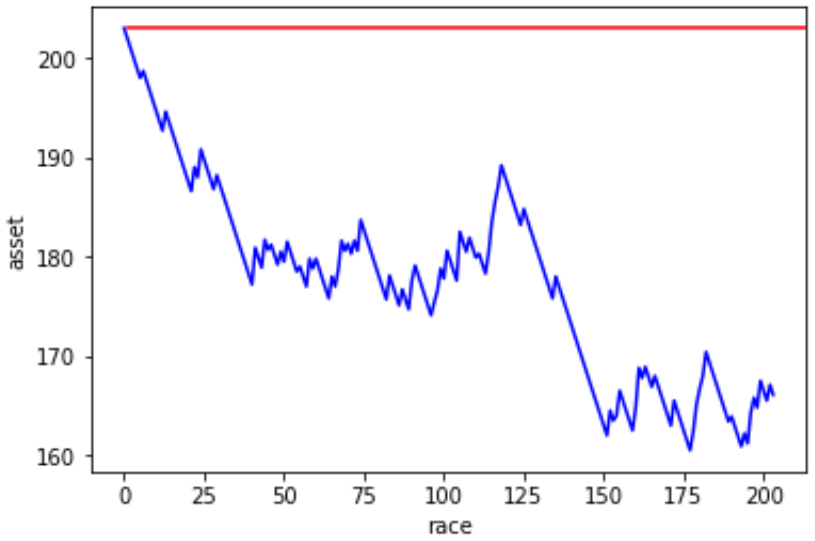
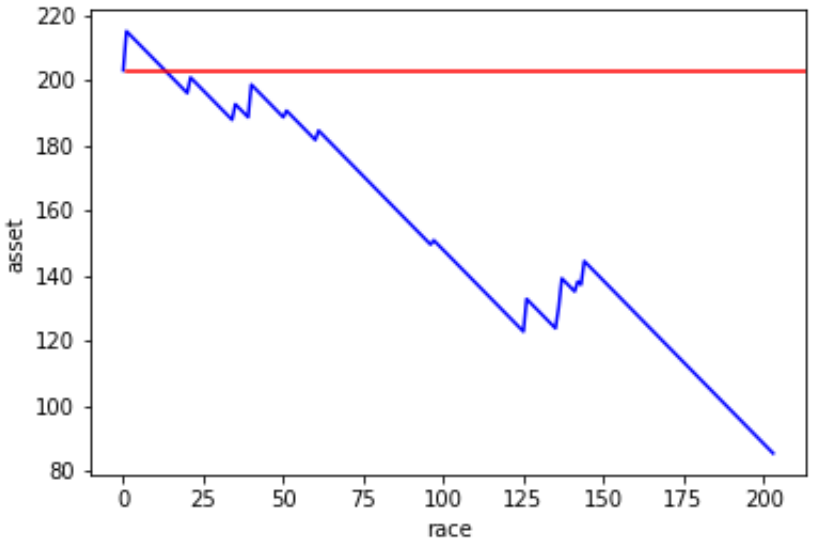
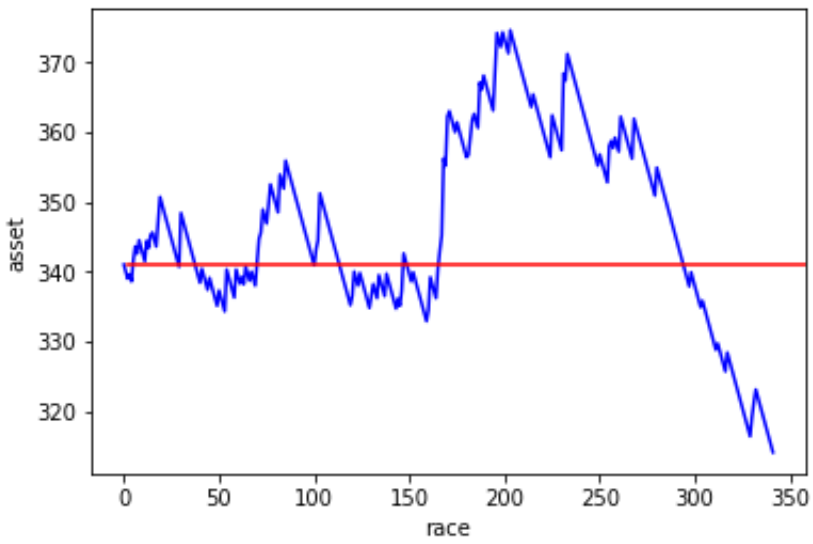
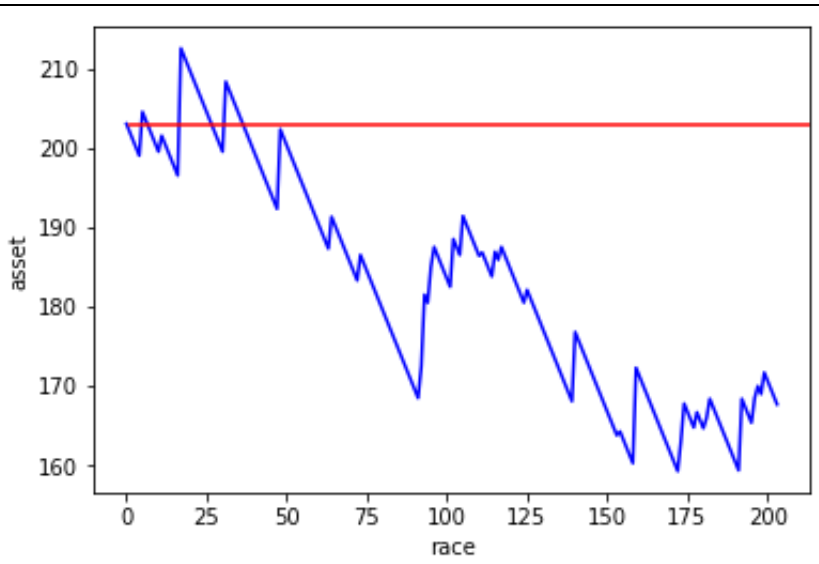
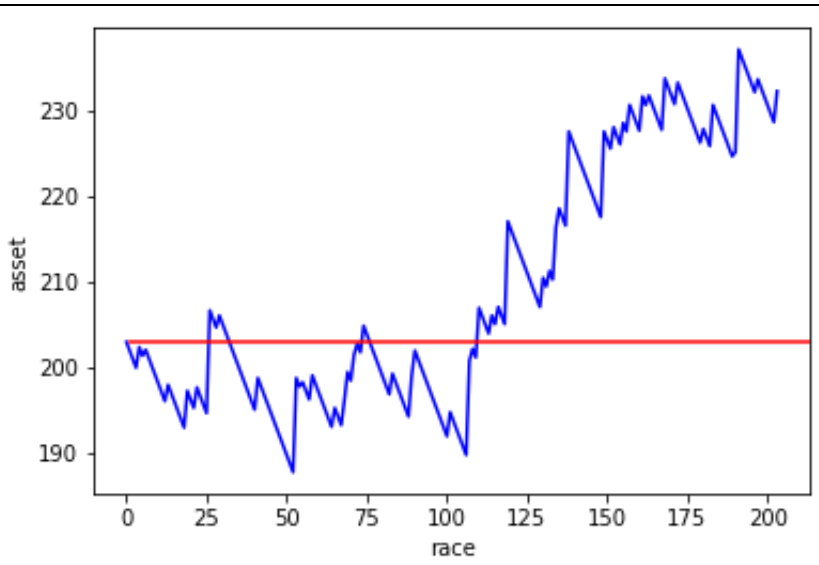
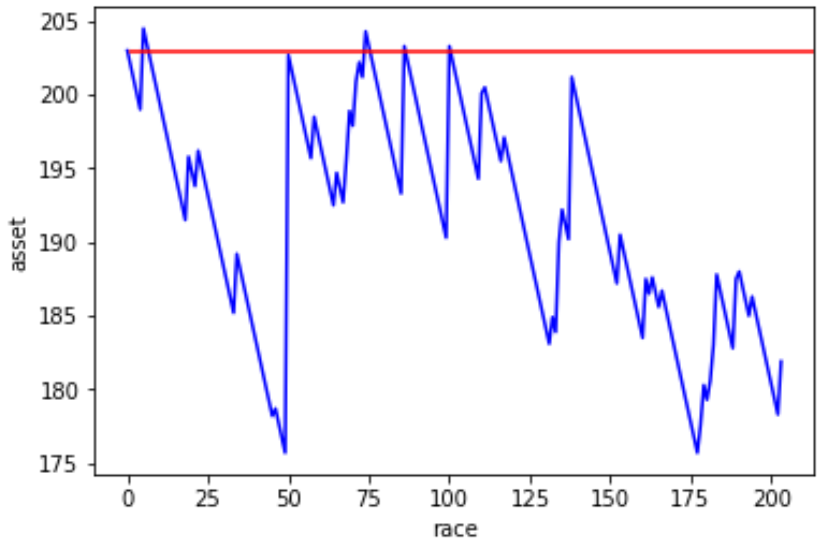
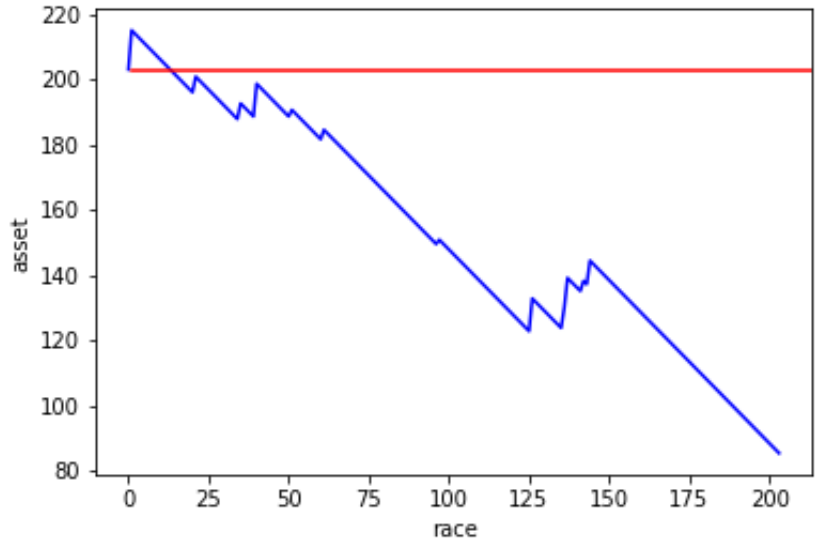
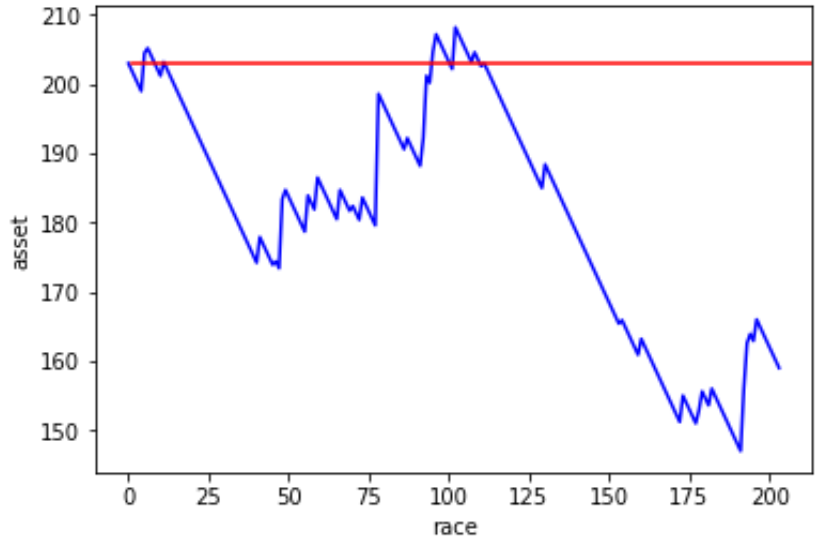
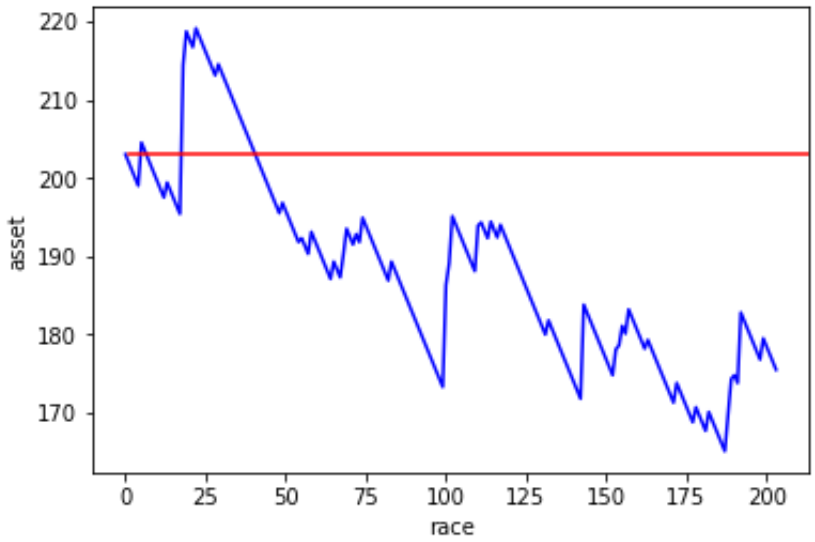
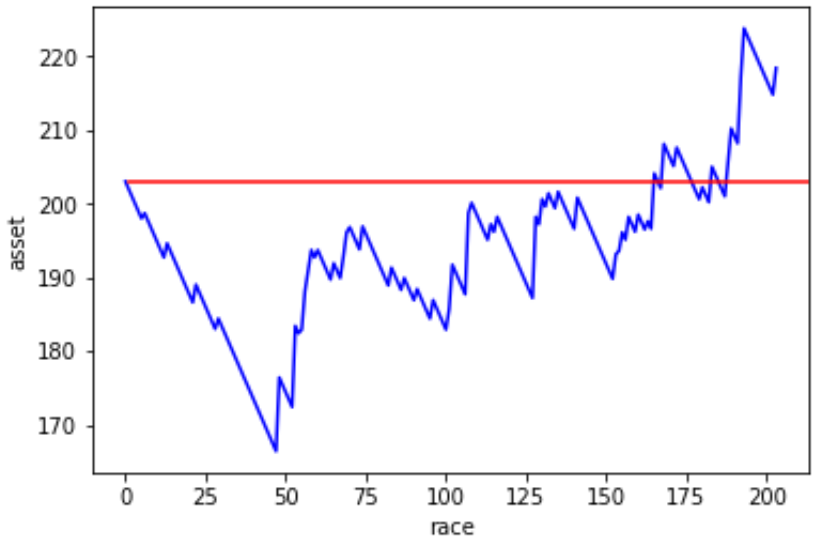
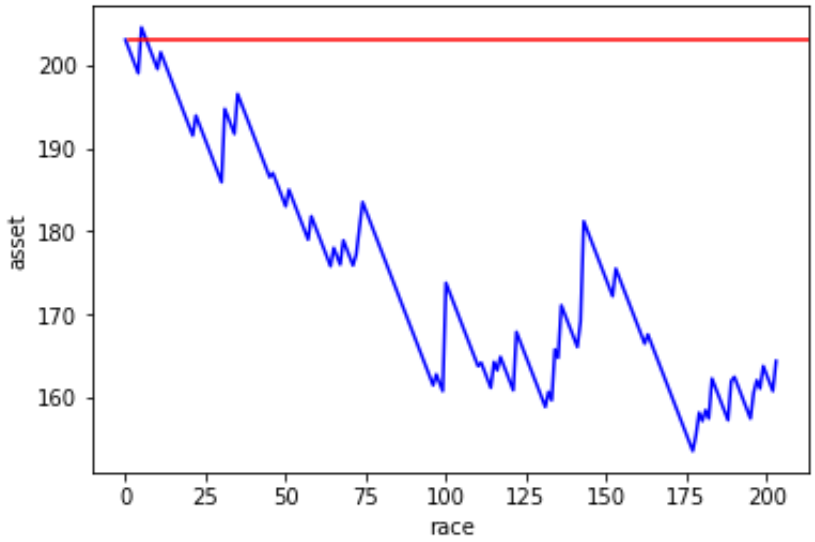
	256	

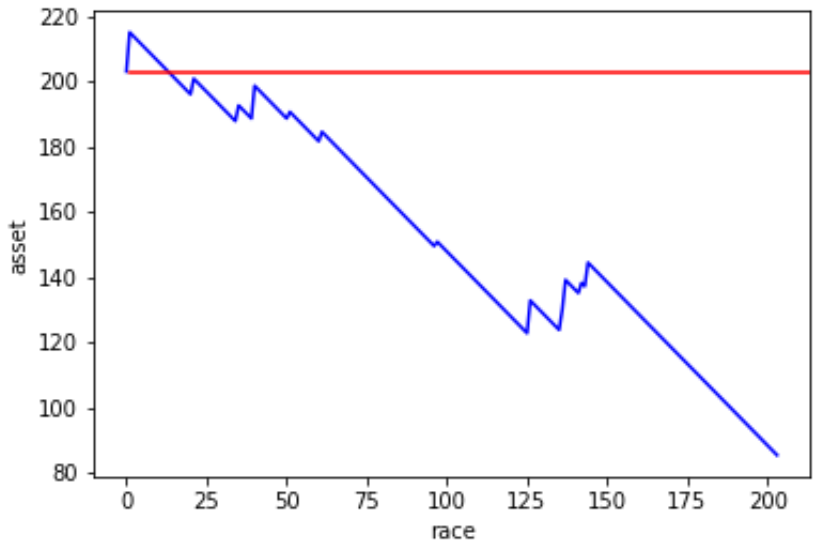
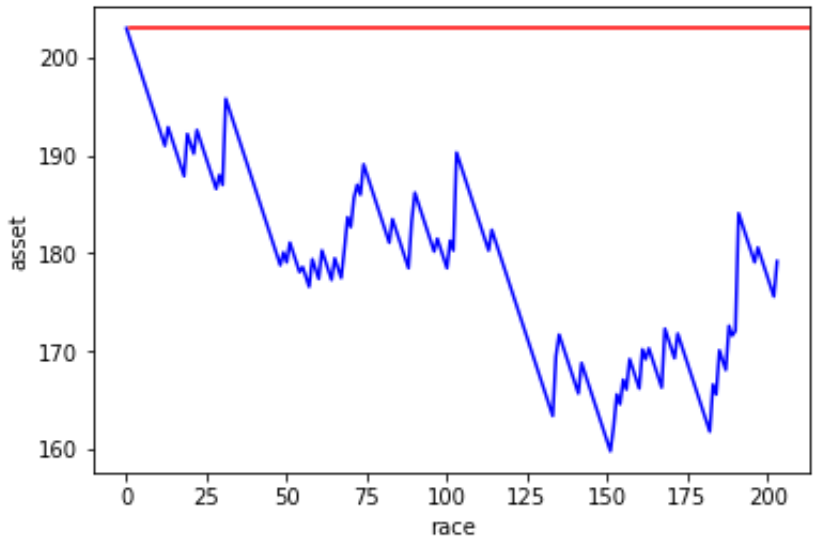
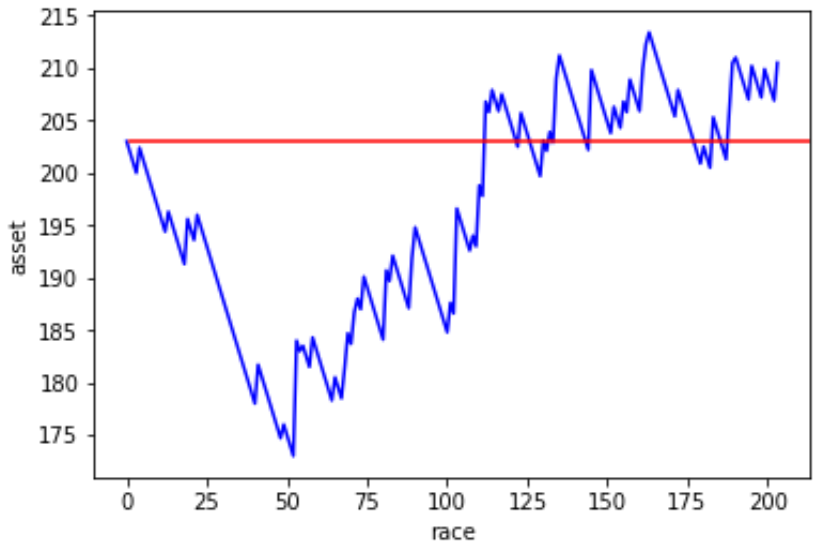
Table 17. Betting curves of 12-horse model with different features and number of neurons

14-horse Model (Tested with 203 14-horse races)		
Feature Set	Neurons	Betting Curve
Public Intelligence	N/A	
All Features	16	

	32	
	64	
	128	

	256	
Without “winodds” Feature	16	
	32	

	64	
	128	
	256	

Without Weather Features	16	
	32	
	64	

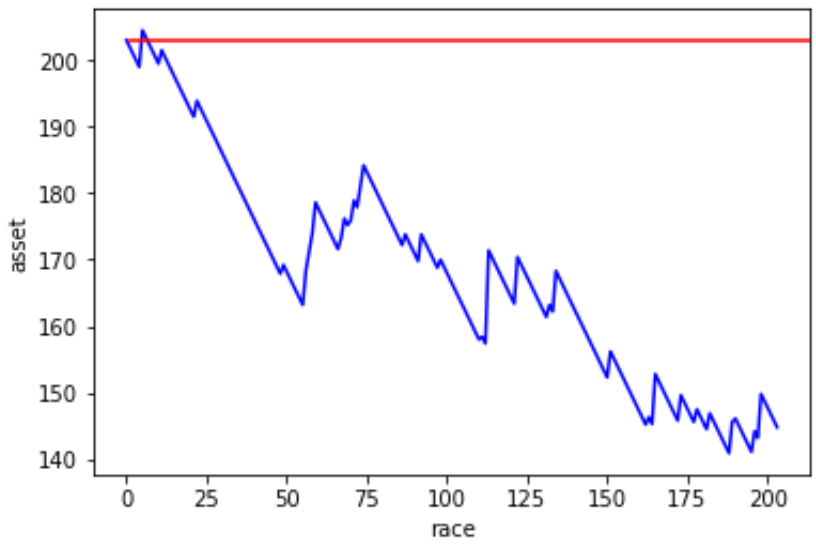
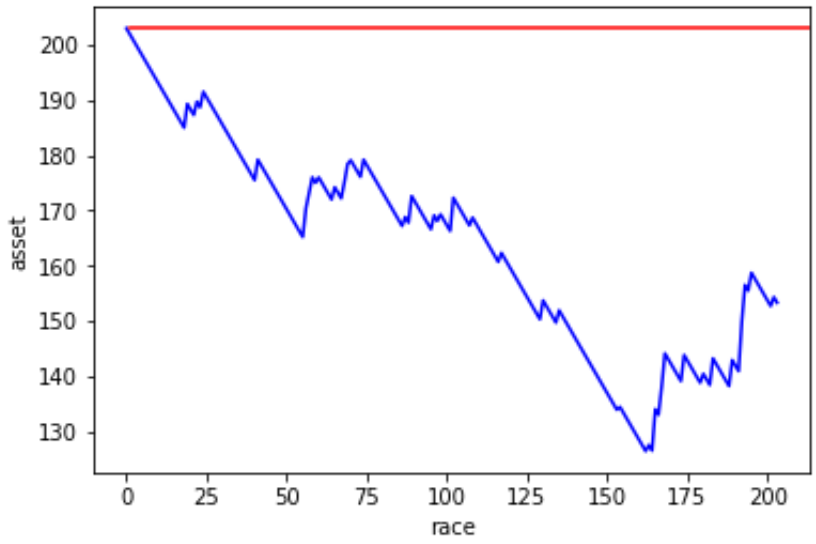
	128	
	256	

Table 18. Betting curves of 14-horse model with different features and number of neurons

8.3 Discussion

8.3.1 Profitability of Horse Racing

From Table 15 and Table 16, we observe that both the 12-horse model and 14-horse model can generate a net profit with certain combination of feature set and number of neurons. The profit is 7.54% for 12-horse model with feature set Without Weather Features at 32 neurons, 14.43%, 7.59% and 3.69% for 14-horse model with feature set All Features at 128 neurons, Without "winodds" Feature at 128 neurons, and Without Weather Features at 64 neurons respectively. This suggests that it is possible to generate profit via horse racing betting and our Bayesian neural network is suitable for predicting horse racing and can generate a net profit.

8.3.2 Optimal Number of Neurons per Layer

For the 12-horse model, best performance for each feature set is obtained at 64 neurons, 64 neurons, and 32 neurons respectively. For the 14-horse model, best performance for each feature set is obtained at 128 neurons, 128 neurons, and 64 neurons respectively. Here we observe two trends, first, the optimal number of neurons for 12-horse model is consistently smaller than that of 14-horse model across all three feature sets, and second, the optimal number of neurons for Without Weather Features is consistently smaller than that of other feature sets across both 12-horse model and 14-horse model.

The reduced optimal number of neurons per layer of 12-horse model compared to 14-horse can be explained by the reduction in number of features as a result of less horses. The number of features per horse is 455 after preprocessing, thus the number of features for 12-horse model is $455 \times 12 = 5460$ while the number of features for 14-horse is $455 \times 14 = 6370$. Therefore, the optimal number of neurons per layer of 12-horse model is smaller than that of 14-horse model.

Similarly, the reduced optimal number of neurons for feature set Without Weather Features can be explained by the reduced complexity of the smaller number of features when weather features are removed. While removing the real valued "winodds" only decrease the features per horse from 455 to 454, removing weather features with many categorical data reduce this number by 64 to 391. This 14% decrease in number of features has led to the reduced optimal number of neurons for feature set Without Weather Features.

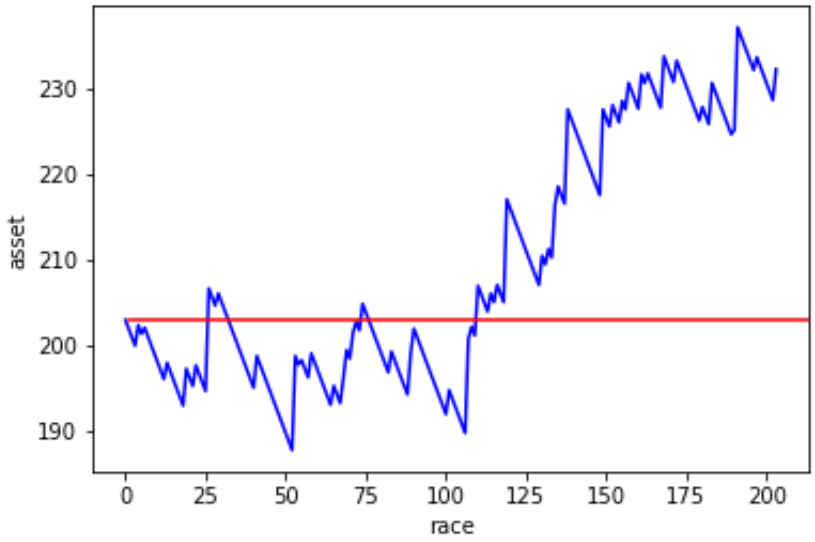
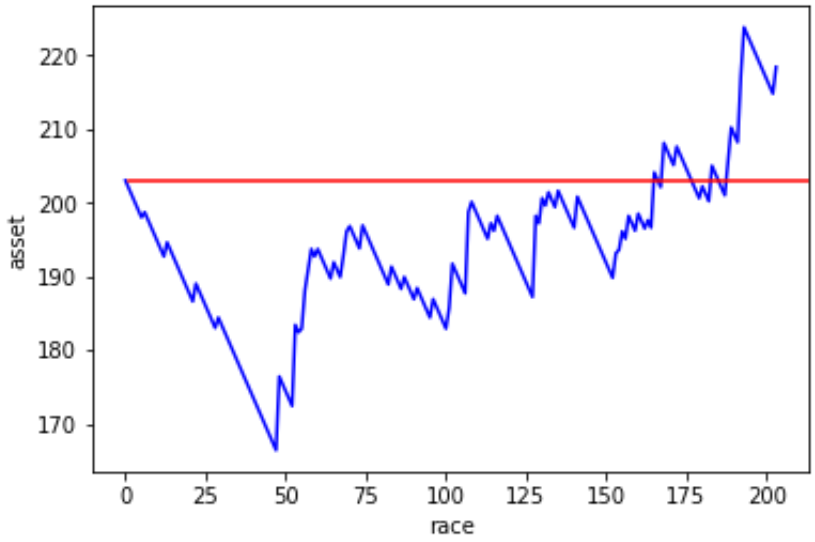
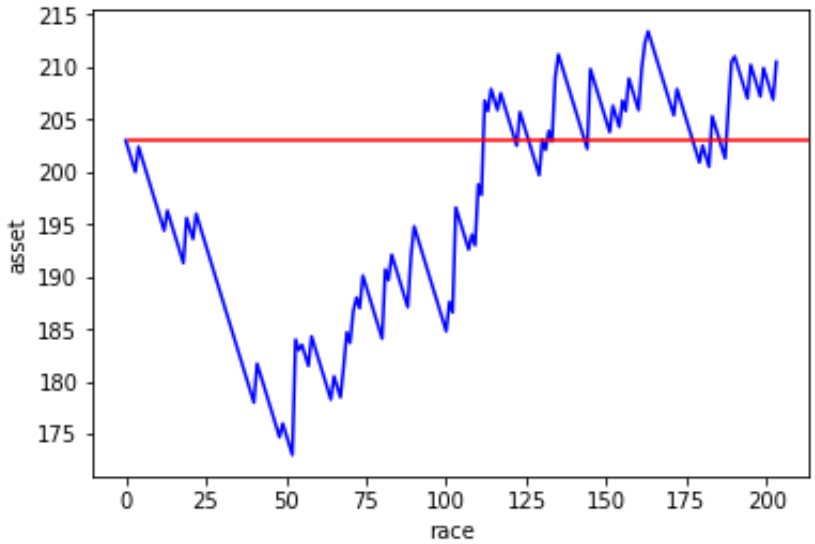
Profitable 14-horse Model (Tested with 203 14-horse races)		
Feature Set	Neurons	Betting Curve
All Features	128	
Without “winodds” Feature	128	
Without Weather Features	64	

Table 19. Betting curves of profitable 14-horse models

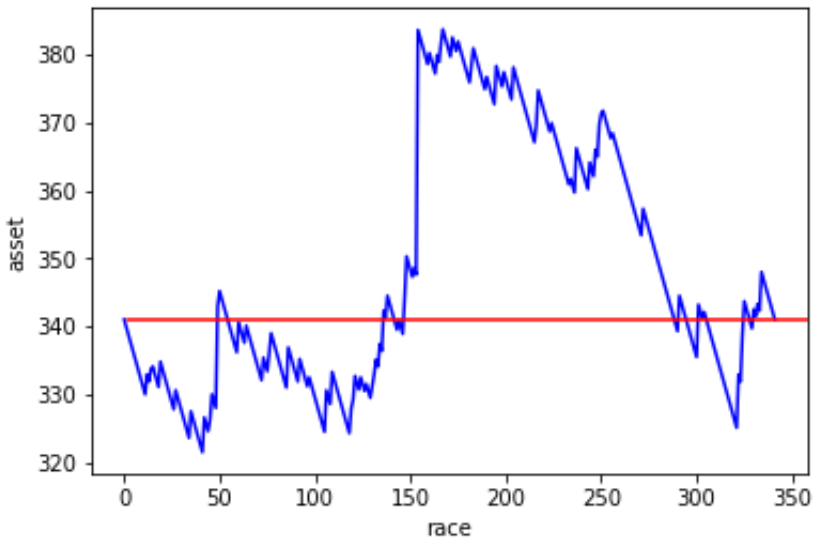
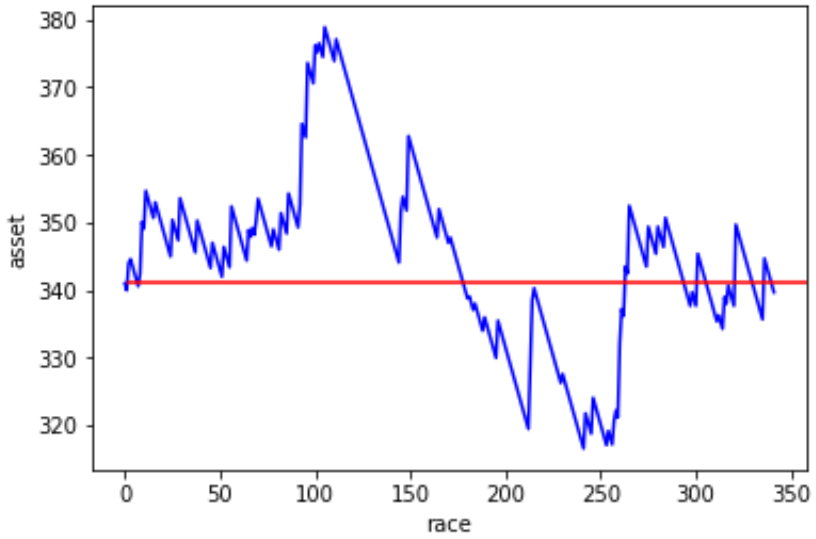
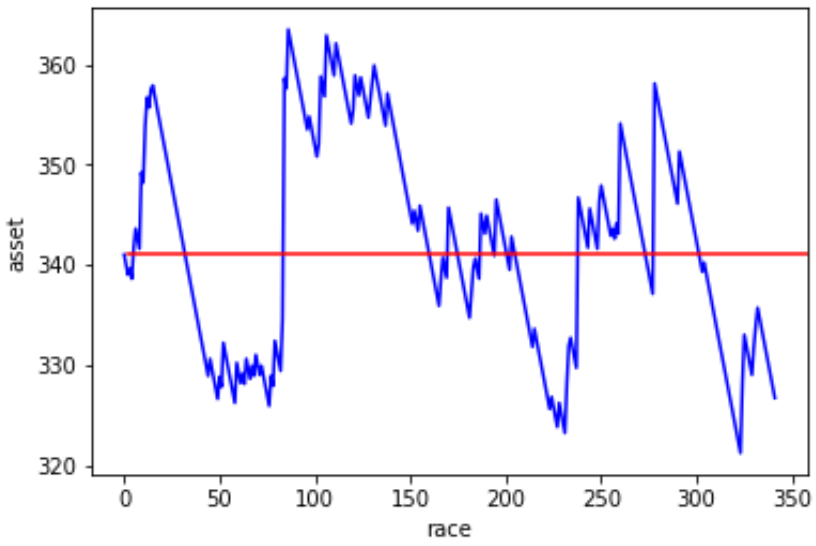
8.3.3 Optimal Feature Set

To better understand the effect of using different feature sets, we extract the three profitable betting curves of 14-horse model from Table 18 to **Error! Reference source not found.**. F rom the table, we can see that the betting behavior using All Features is different from that of Without "winodds" Feature and Without Weather Features and is more stable without exhibiting large losses throughout the betting in the testing year, while removing either "winodds" or weather features results in a large initial drop in asset. This suggests that the combined knowledge of "winodds" or weather are important in predicting horse racing.

While this may contradict with the result for 12-horse model, where the only profitable feature set is Without Weather Features, we attribute this result to the influence of other factors in data augmentation. To verify our hypothesis, we rerun the experiment for 12-horse model without data cropping of 13 or 14-horse race and only use 12-horse races for training. Due to time and resources constraint, we only rerun the experiment with the optimal configurations of 12-horse model and 14-horse model. The results are shown in Table 20 and Table 21. Without data augmentation, the performance of using all features are the best in terms of both accuracy and net return. Therefore, we can conclude that using All Features are the best when there is no data augmentation. This seems to suggest that data augmentation by cropping influences the performance of model under different feature set. Further investigation of influence of data augmentation must be done in order to explain how data augmentation influence the performance of model under different feature set.

12-horse Model without Data Augmentation (Tested with 341 12-horse races)					
Feature Set	Neurons	Accuracy (%)	Net Return	Return/Bet (%)	Profit?
All Features	64	20.53	0.0	0.0	No
	128	17.01	–1.3	–0.38	No
Without "winodds" Feature	64	19.65	–14.3	–4.19	No
	128	14.66	–39.7	–11.64	No
Without Weather Features	32	19.35	–64.7	–18.97	No
	64	15.84	–87.3	–25.60	No

Table 20. Performance of 12-horse model without data augmentation

<table>
<tr><td colspan="3">12-horse Model without Data Augmentation (Tested with 203 14-horse races)</td></tr>
<tr><td>Feature Set</td><td>Neurons</td><td>Betting Curve</td></tr>
<tr><td rowspan="2">All Features</td><td>64</td><td> 
</td></tr>
<tr><td>128</td><td> 
</td></tr>
<tr><td>Without “winodds” Feature</td><td>64</td><td> 
</td></tr>
</table>

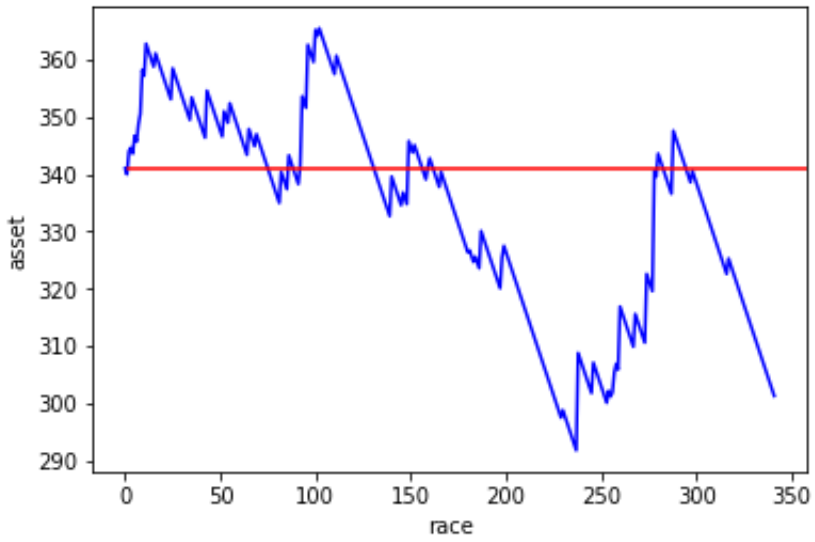
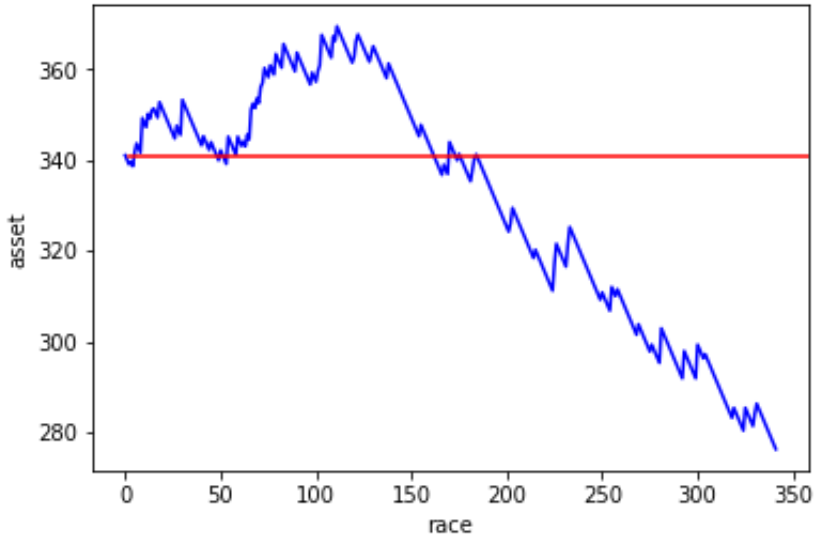
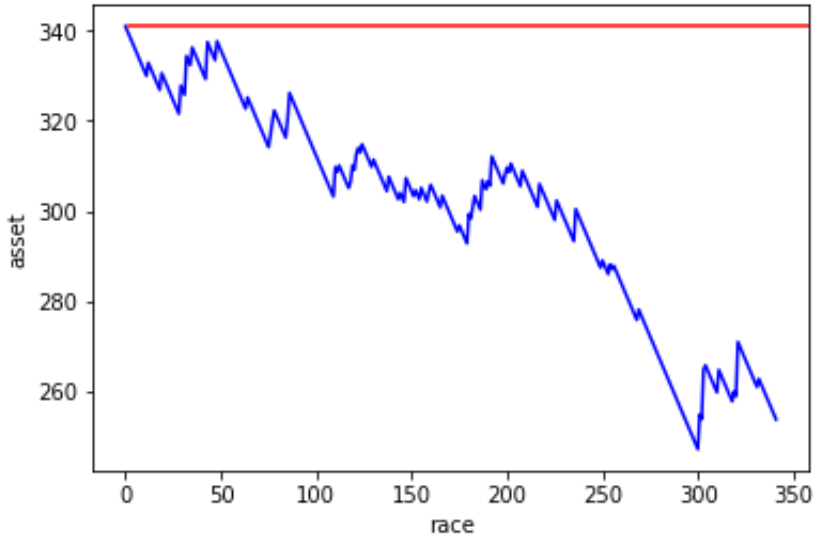
	128	
Without Weather Features	32	
	64	

Table 21. Betting curves of 12-horse model without data augmentation

8.4 Kelly Betting

The fixed betting in the previous sections may not be the optimal betting strategy of a rational being. Indeed, experienced human betters may not bet fixed amounts on every race even when they hold different confidences about the races [35]. To incorporate confidence into betting, we employ Kelly betting on each model. Kelly betting is a formula for bet sizing that leads to optimal wealth increase in the long run as the number of bets goes to infinity [35]. It sets the bets size by maximizing the expected logarithm of wealth which is equivalent to maximizing the expected geometric growth rate.

For example, let p be the probability of winning, and b be the return per unit bet, which is the amount won per unit bet on top of getting the bet amount back. Also, we denote the current amount of asset to be A. Then, the Kelly bet is

$$f = A \times \frac{p(b+1)-1}{b}$$

In our experiments, we take the output of the neural network as the probability of winning of each horse. The sum of this output is 1, which is consistent to be interpreted as probability. To give a fair comparison to fixed betting, the initial asset is set to equal to 341 for 12-horse model and 203 for 14-horse model. The results are shown on the next page.

8.4.1 Discussion

From the results, most of our models lost all the asset when utilizing Kelly betting, even when it can generate a profit using fixed betting. One possible explanation is that while our models are comparatively accurate in predicting the winning horse, they are over confidence in its prediction and therefore lost all of their asset.

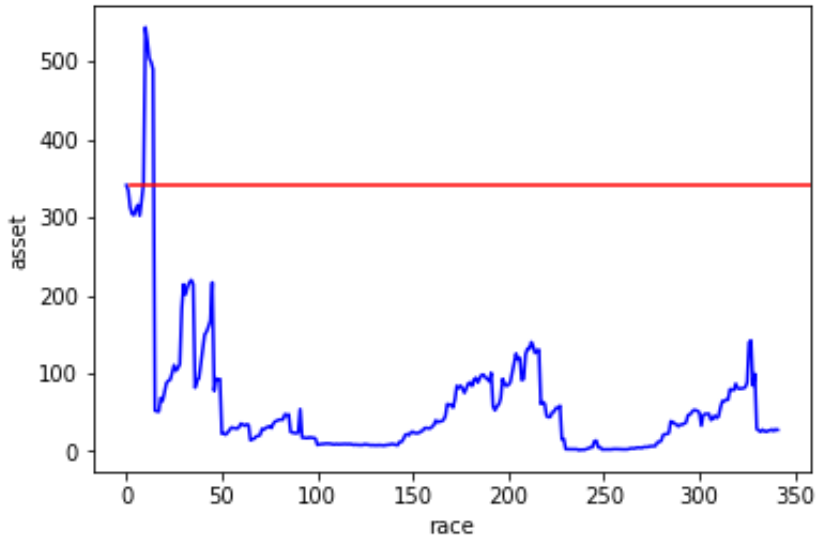
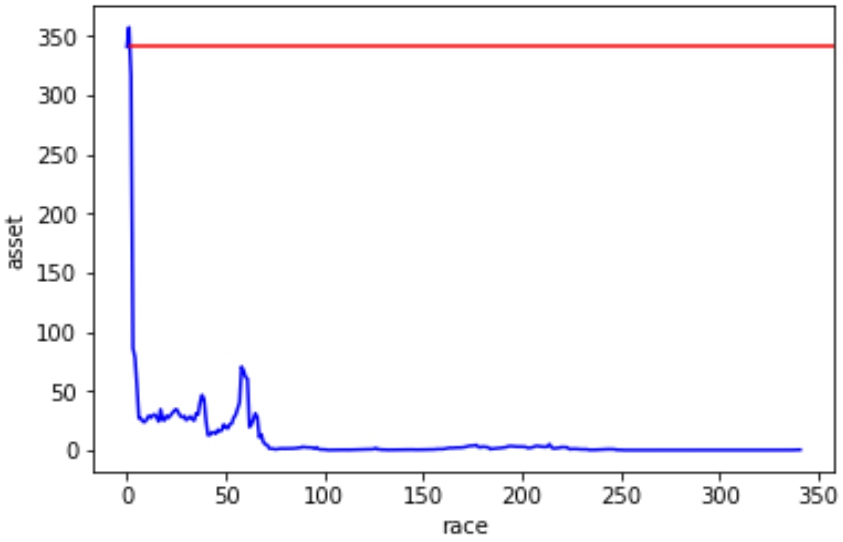
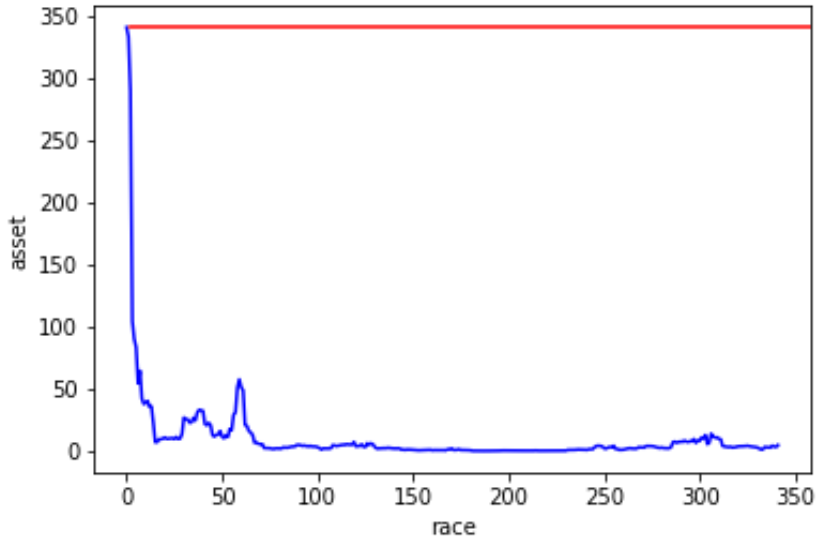
One notable exception is the 14-horse model with 64 neurons using Without “winodds” Feature feature set. This model obtained a 403.22% net gain of 818.5 and the highest amount reached in the process is over 35000. However, without additional testing data, we are unable to conclude whether this model is about to make a profit in the long run.

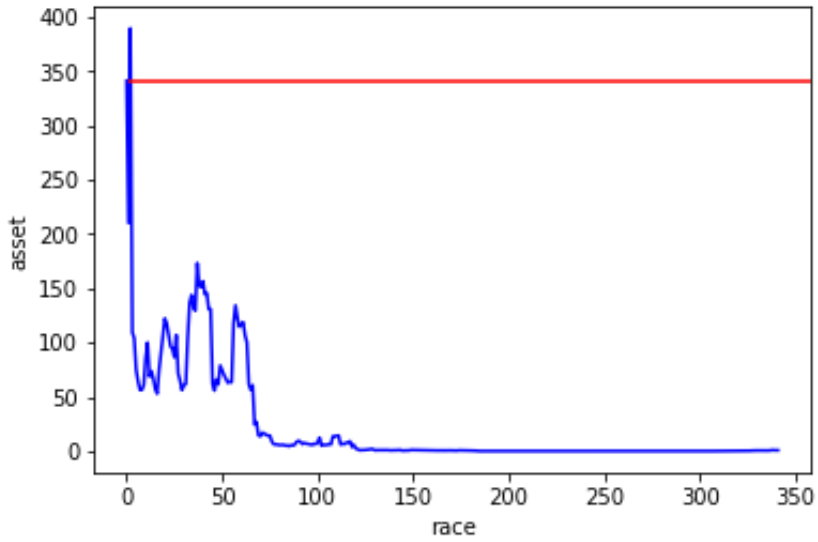
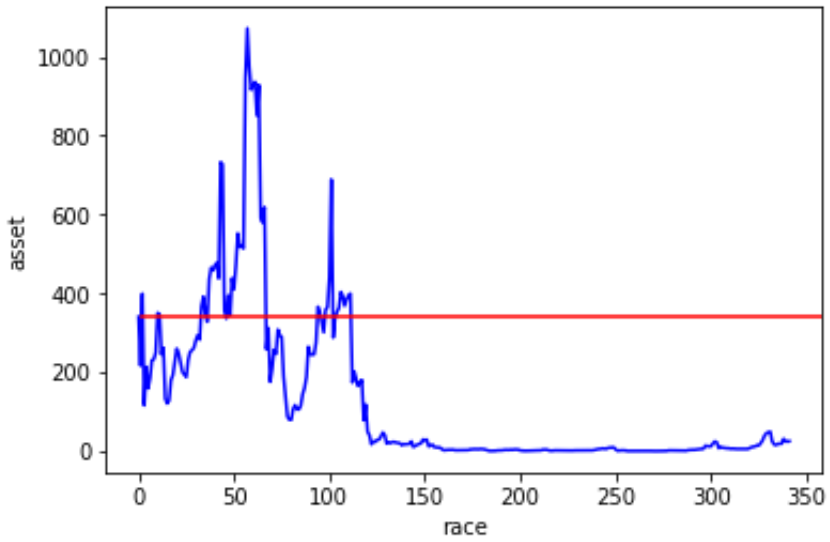
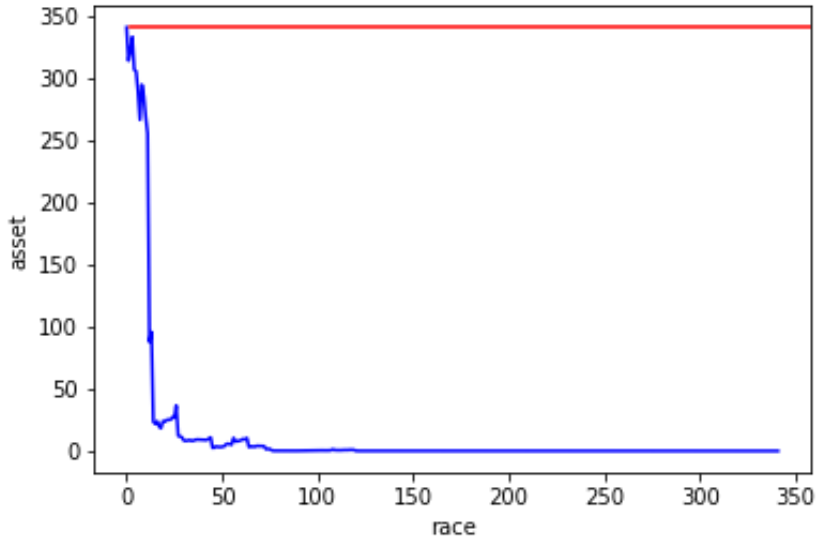
12-horse Model (Tested with 341 12-horse races)					
Feature Set	Neurons	Accuracy (%)	Net Return	Return/Bet (%)	Profit?
Public Intelligence	N/A	22.87	–114.5	–33.58	No
All Features	16	7.62	–313.6	–91.95	No
	32	17.01	–340.6	–99.90	No
	64	22.58	–336.3	–98.61	No
	128	17.60	–340.1	–99.75	No
	256	18.18	–315.3	–92.50	No
Without "winodds" Feature	16	8.80	–341.0	–100	No
	32	8.80	–341.0	–100	No
	64	20.82	–341.0	–100	No
	128	16.42	–340.2	–99.77	No
	256	17.60	–341.0	–100	No
Without Weather Features	16	9.68	–315.7	–92.57	No
	32	22.29	–314.9	–92.34	No
	64	19.35	–332.4	–97.48	No
	128	17.30	–338.6	–99.30	No
	256	17.30	–341.0	–100	No

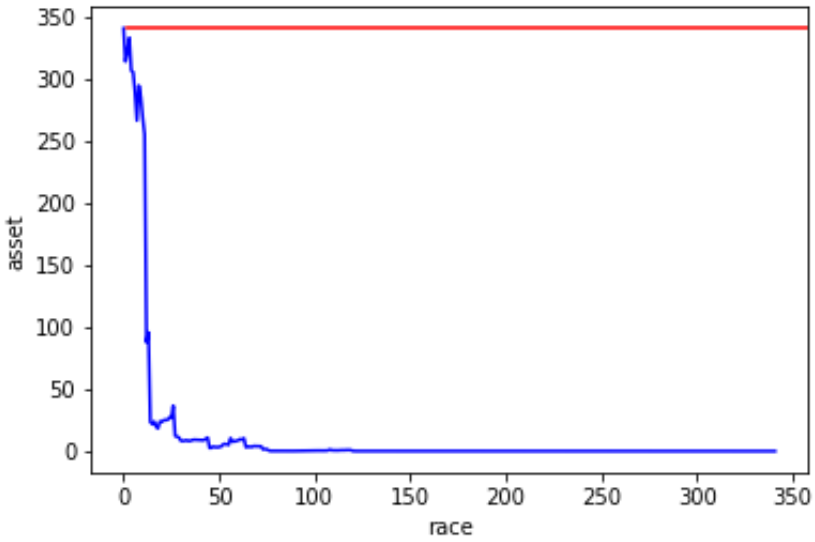
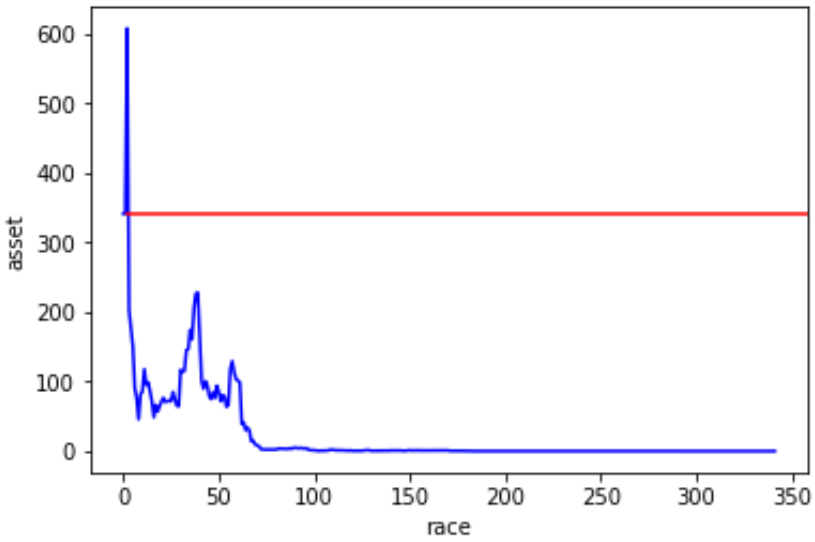
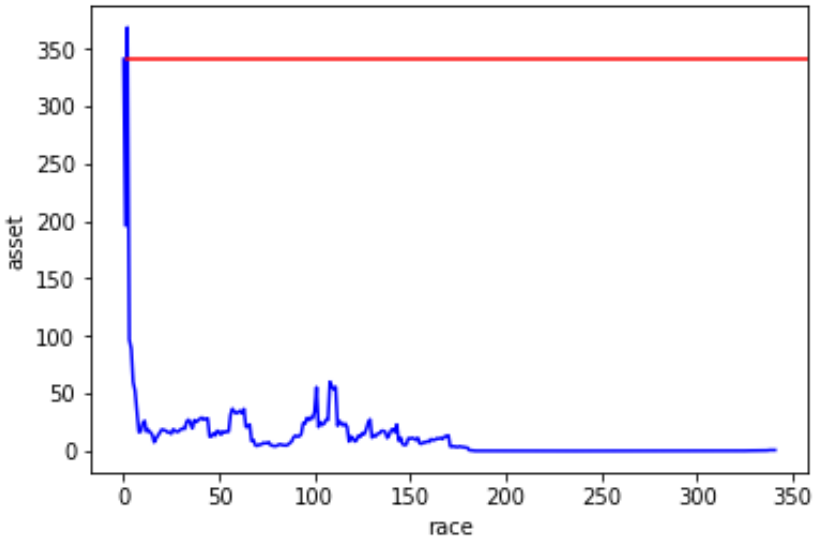
Table 22. Kelly betting performance of 12-horse model with different features and number of neurons

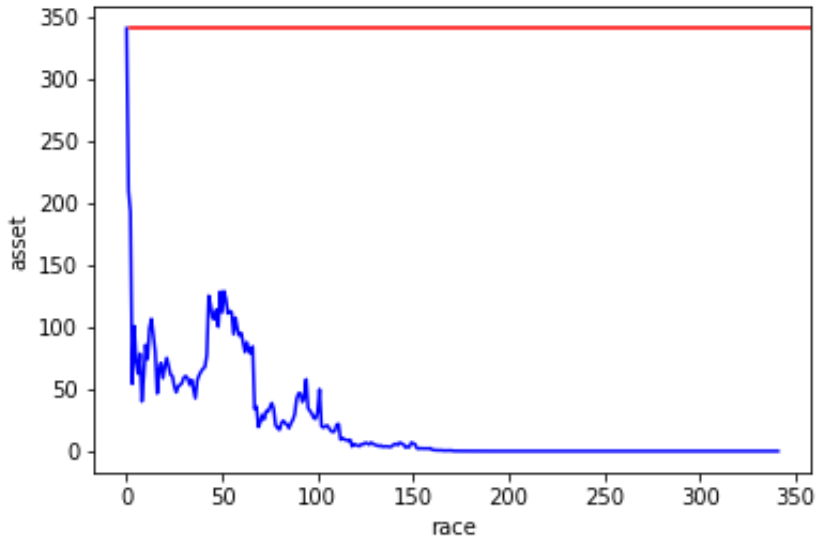
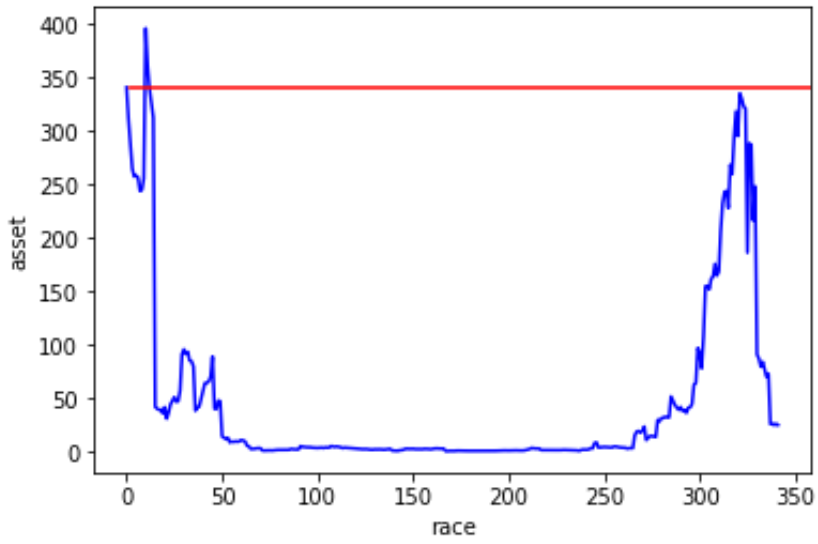
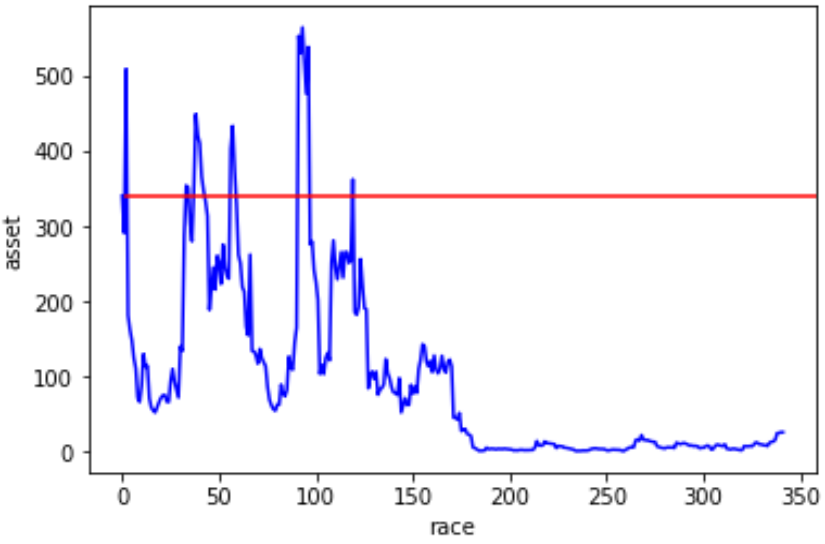
14-horse Model (Tested with 203 14-horse races)					
Feature Set	Neurons	Accuracy (%)	Net Return	Return/Bet (%)	Profit
Public Intelligence	N/A	27.59	–36.9	–18.18	No
All Features	16	5.91	–203.0	–99.98	No
	32	16.75	–202.7	–99.86	No
	64	14.29	818.5	403.22	Yes
	128	22.66	–201.6	–99.29	No
	256	17.24	–202.6	–99.79	No
Without "winodds" Feature	16	9.85	–203.0	–99.98	No
	32	15.76	–203.0	–99.99	No
	64	18.23	–203.0	–100	No
	128	23.15	–203.0	–100	No
	256	17.73	–203.0	–99.98	No
Without Weather Features	16	5.91	–203.0	–99.98	No
	32	20.20	–202.5	–99.75	No
	64	23.15	–202.9	–99.95	No
	128	15.76	–202.8	–99.89	No
	256	17.24	–202.7	–99.86	No

Table 23. Kelly betting performance of 14-horse model with different features and number of neurons

12-horse Model (Tested with 341 12-horse races)		
Feature Set	Neurons	Kelly Betting Curve
All Features	16	
	32	
	64	

	128	
	256	
Without “winodds” Feature	16	

	32	 asset race 0 50 100 150 200 250 300 350
	64	 asset race 0 100 200 300 400 500 600
	128	 asset race 0 50 100 150 200 250 300 350

	256	
Without Weather Features	16	
	32	

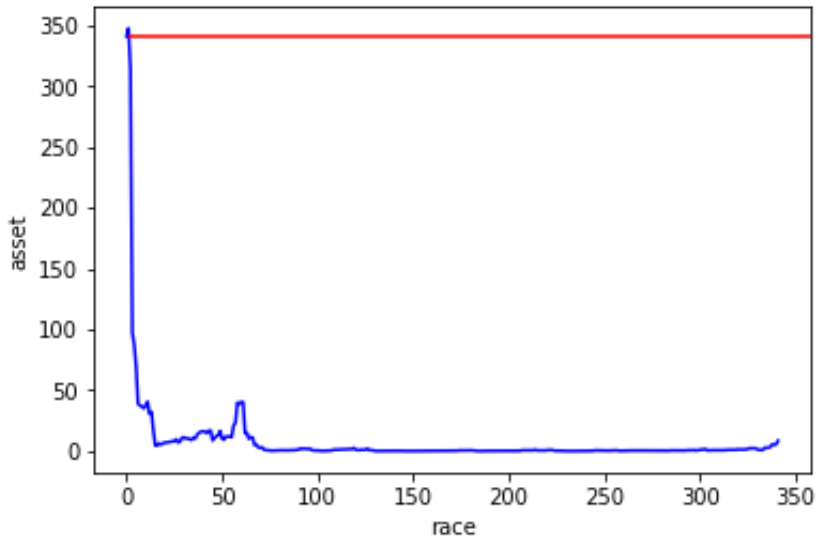
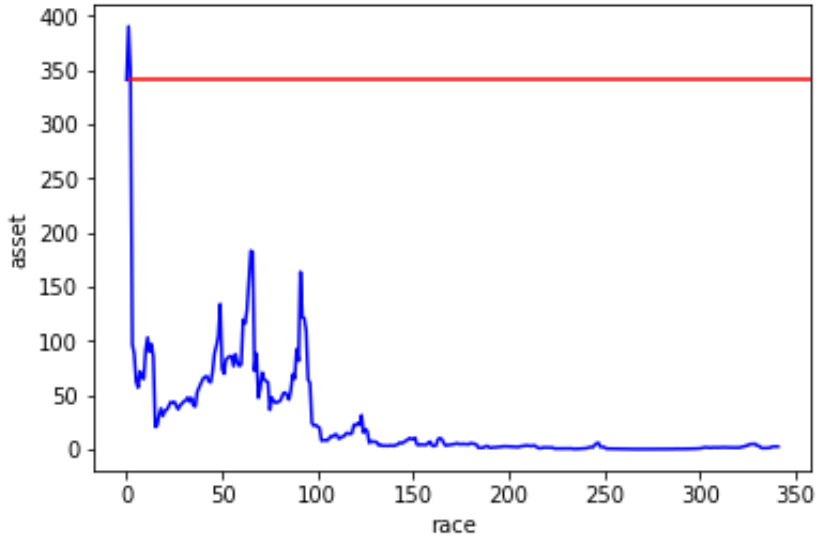
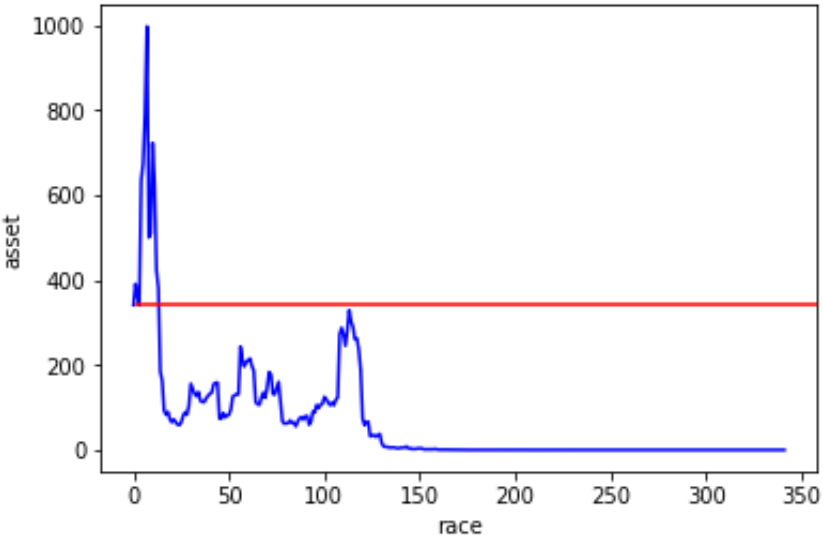
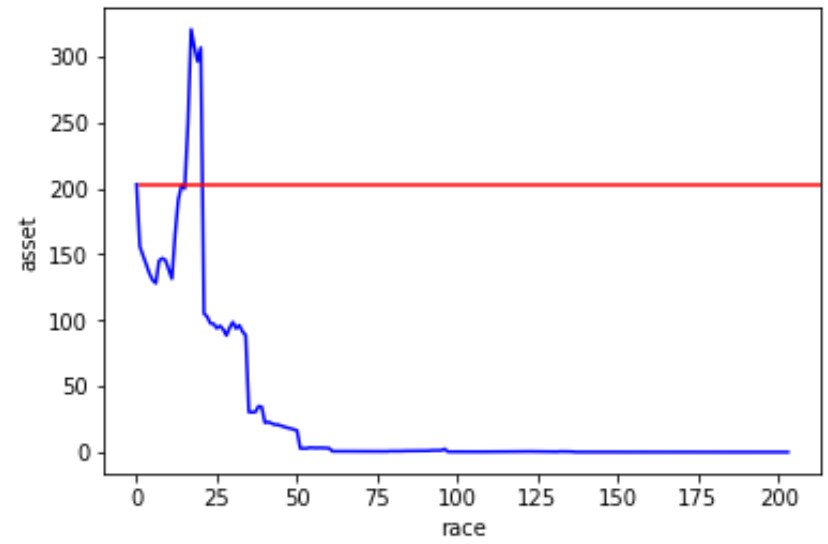
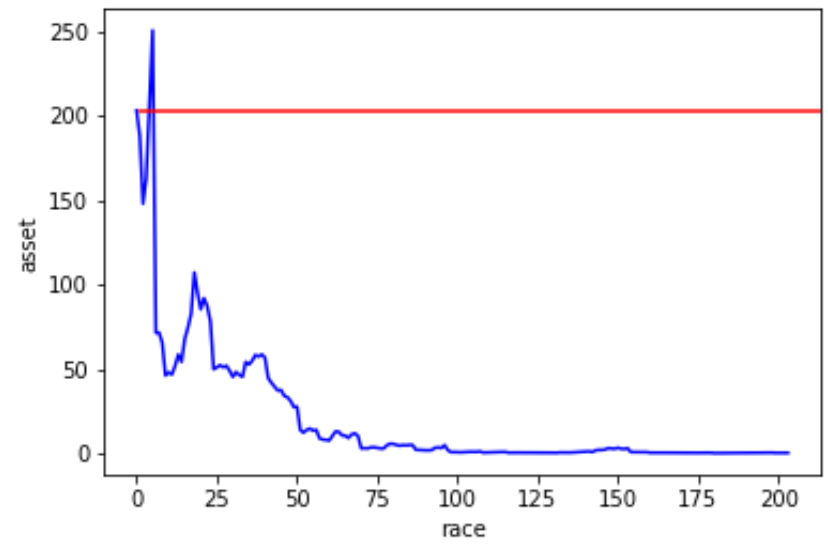
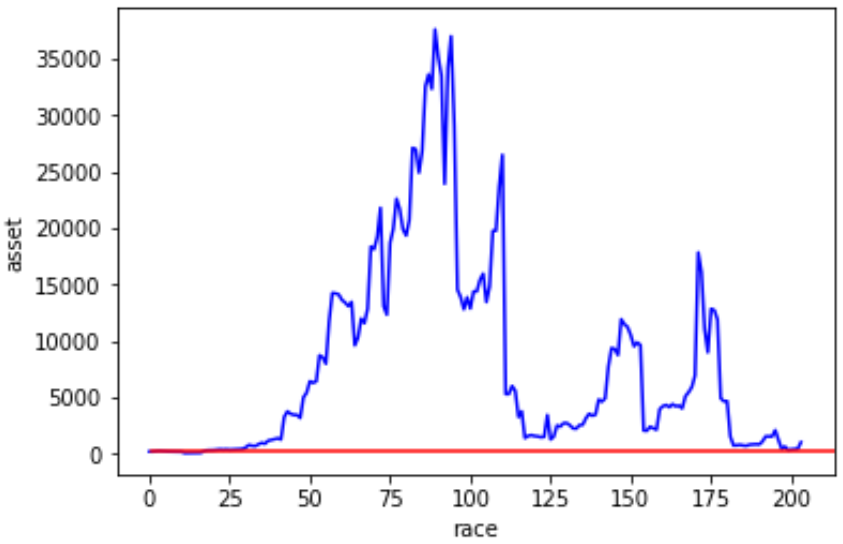
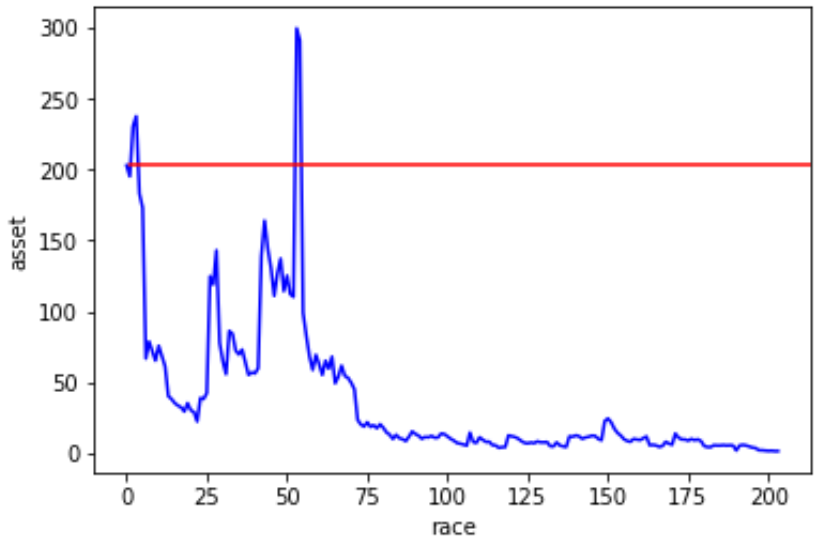
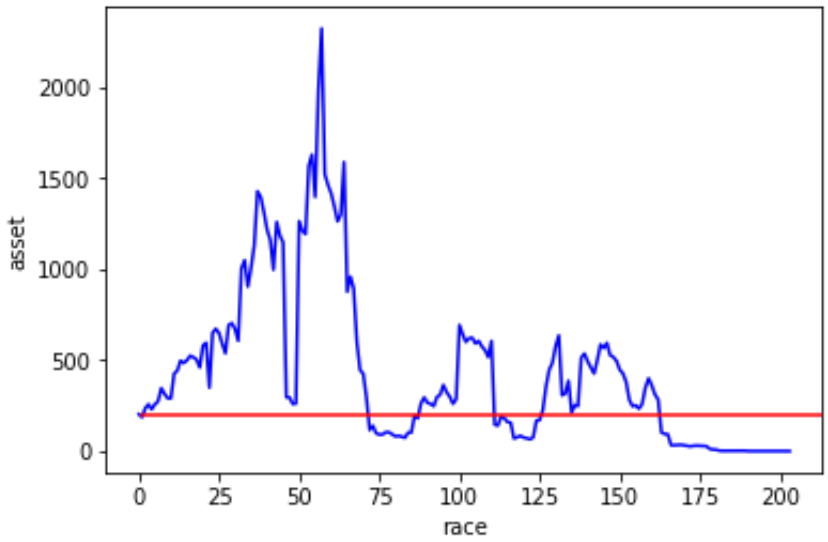
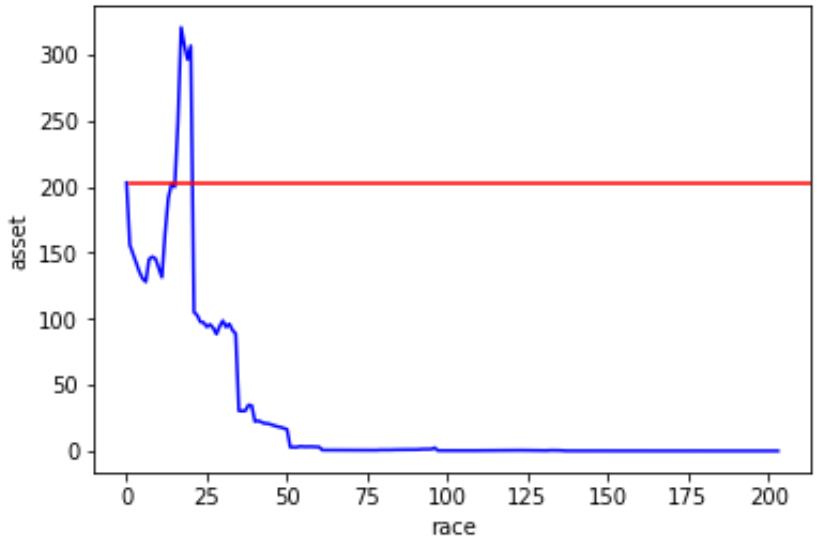
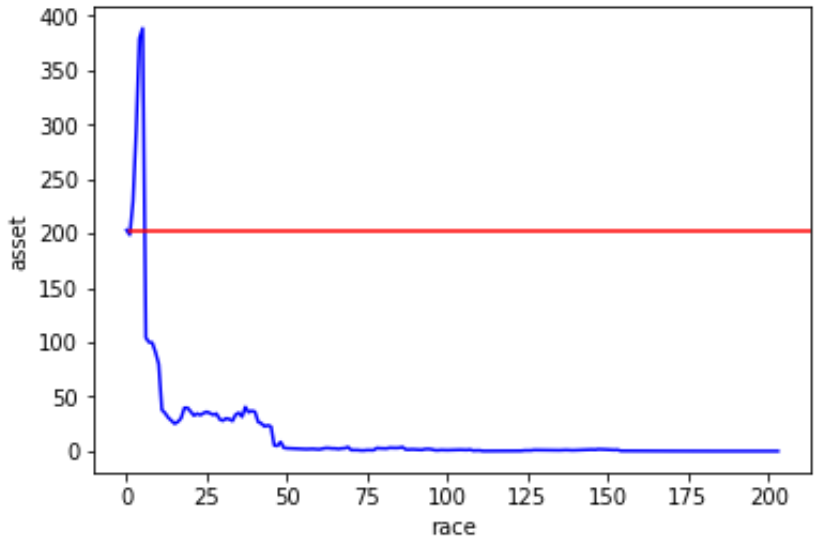
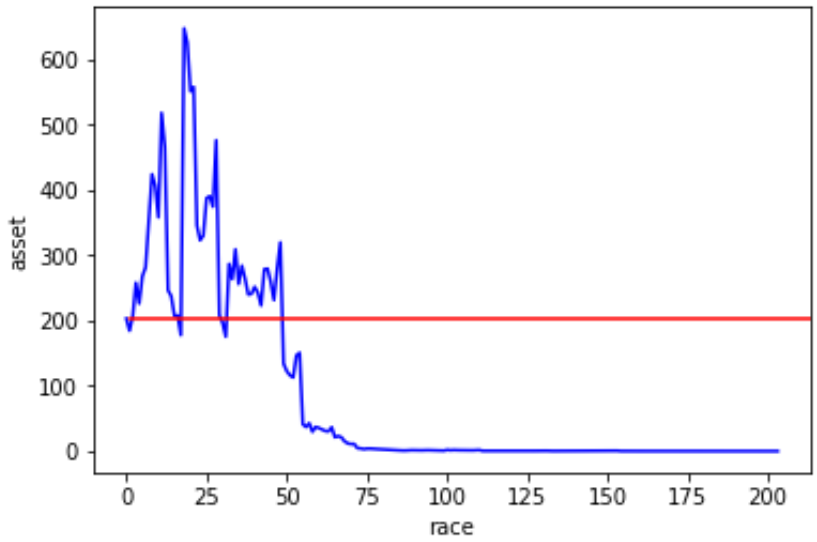
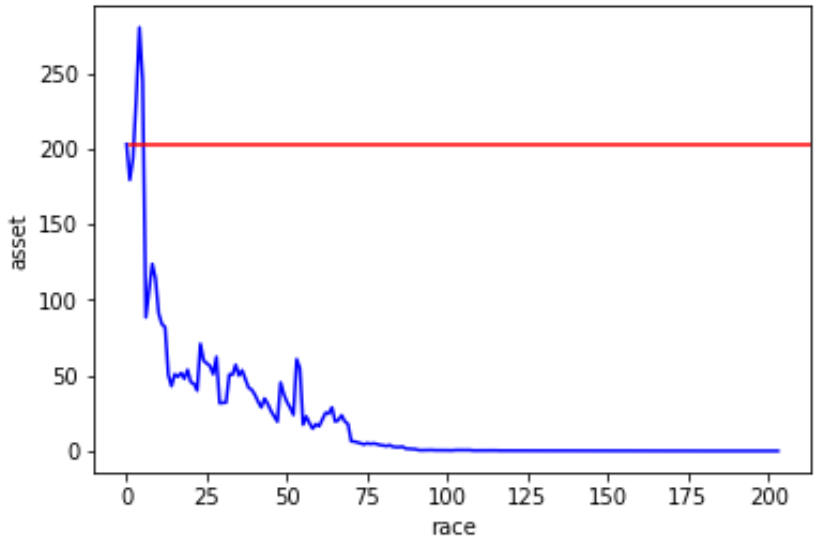
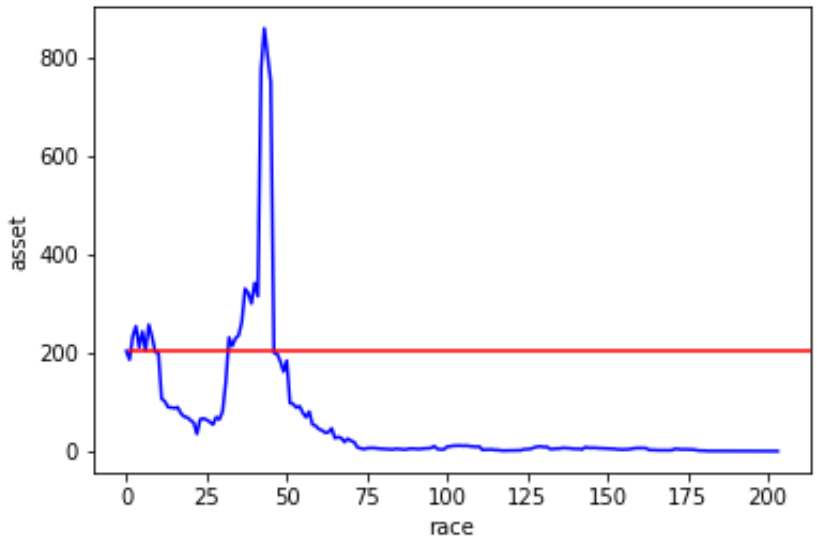
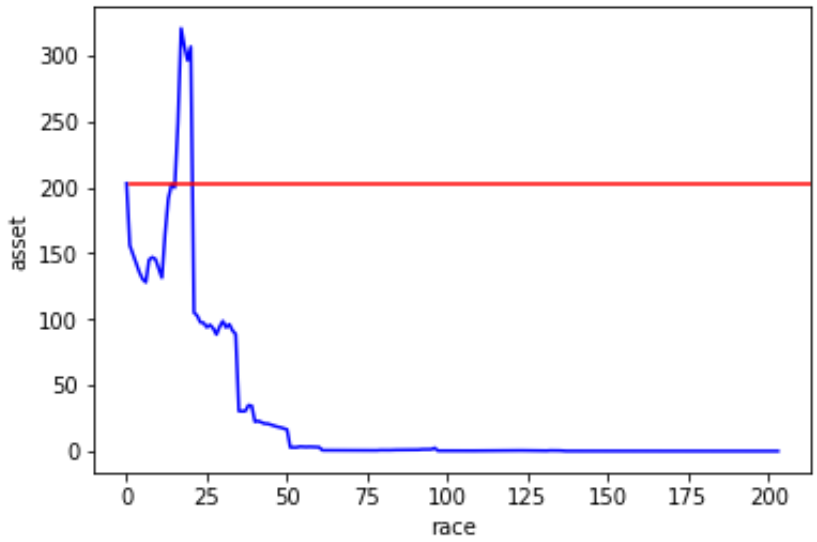
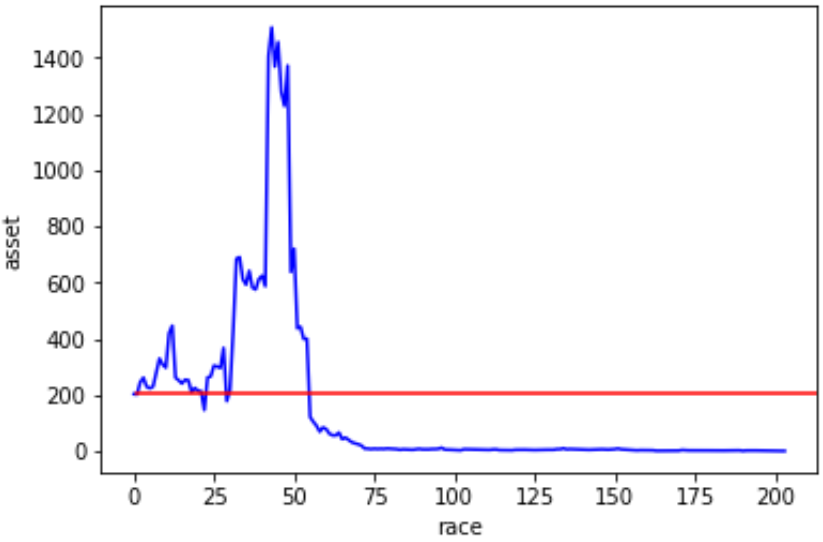
	64	
	128	
	256	

Table 24. Kelly betting curves of 12-horse model with different features and number of neurons

14-horse Model (Tested with 203 14-horse races)		
Feature Set	Neurons	KellyBetting Curve
All Features	16	
	32	
	64	

	128	
	256	
Without “winodds” Feature	16	

	32	
	64	
	128	

	256	
Without Weather Features	16	
	32	

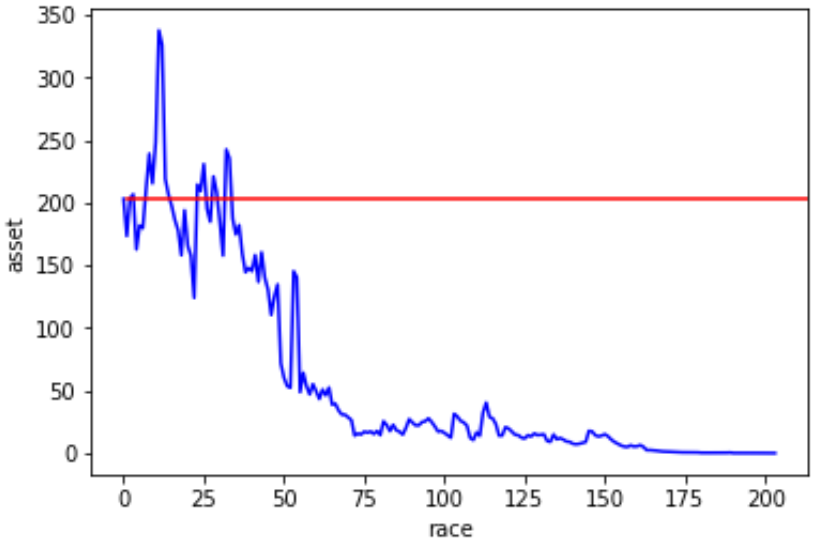
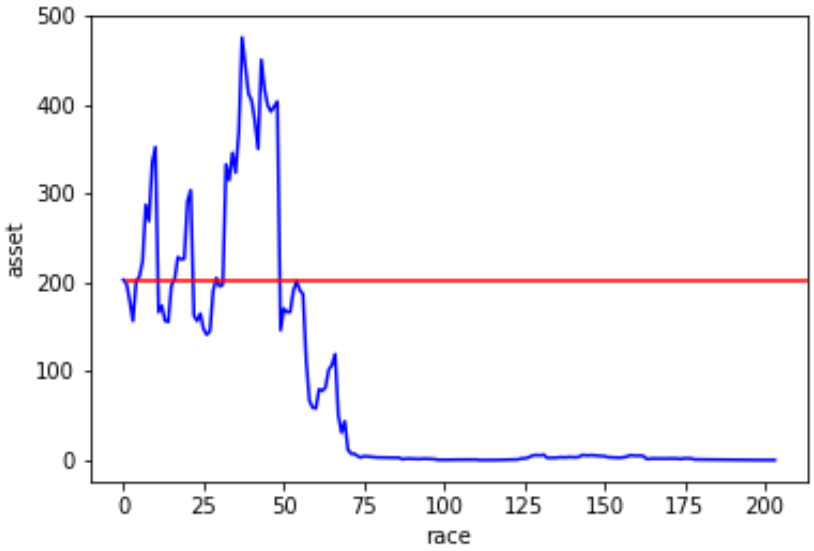
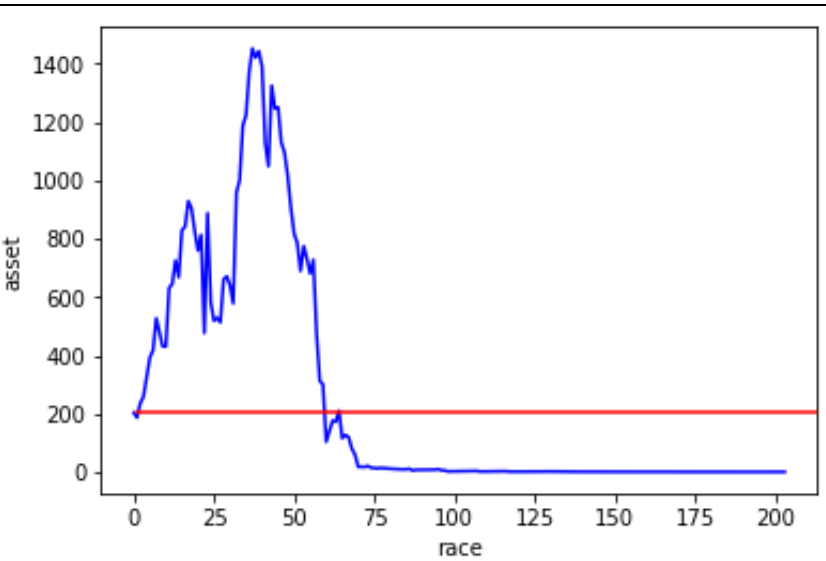
	64	
	128	
	256	

Table 25. Kelly betting curves of 14-horse model with different features and number of neurons

8.5 Comparison with Related Works

In this project, we used a similar dataset as Liu and Wang [8] with Hong Kong races from 2011 to 2018. The data of the same time period from 2011 to 2017 are used as training set, while the testing data set are extended to races of the whole year 2018 instead of only from January to April to minimize testing noise and provide a better measure of the model performance. The use of training data on the same period allows a direct comparison between our work and that of Liu and Wang [8].

The best model of Liu and Wang [8] can achieve 24.51% in win accuracy and result in a net loss of 25.78%. Our best 14-horse model have win accuracy of 22.66% and a net gain of 14.43%, and 12-horse model have win accuracy of 22.29% and a net gain of 7.54%. While our model does not exceed that of [8] in terms of accuracy, we note that the number of horses in a race of their testing data ranges from 6 horses to 14 horses, thereby increasing the testing accuracy as a result of less uncertainty due to less horses in a race, while our testing data is only composed of 12-horse races and 14-horse races. A fair comparison of accuracy cannot be done without setting the same number of horses in the testing data for [8] and our model. In terms of net gain however, our model can generate a net profit while that of Liu and Wang [8] cannot.

Apart from Liu and Wang [8], Cheng and Lau [7] have also used neural networks for horse racing prediction. Compared to our work and Liu and Wang [8], Cheng and Lau [7] used a larger 16–year dataset of Hong Kong races from 2001 to 2016. Their neural network model achieved win accuracy of 21.42%, and when the threshold is not used, the model result in a loss of over 20%. Our best 14-horse model have win accuracy of 22.66% and a net gain of 14.43%, and 12-horse model have win accuracy of 22.29% and a net gain of 7.54%. Therefore, our model performed better than that of [7] in terms of both accuracy and net gain.

While our Bayesian neural network model does not perform significantly better than neural network models of Cheng and Lau [7], Liu and Wang [8] in term of accuracy, we are able to perform better in terms of net gain. We partly attribute this to the difference of objectives between Bayesian neural network and neural network, the first attempts to infer the true posterior probability of the network weights, while the latter optimizes directly on accuracy.

In the last term, we have used a similar Bayesian neural network model as current model but for predicting the place of each individual horse [9], and achieved 25.92% accuracy and net

loss of –20.09%. Our best 14-horse model have win accuracy of 22.66% and a net gain of 14.43%, and 12-horse model have win accuracy of 22.29% and a net gain of 7.54%. Similar to [8] the test data we used last term [9] has number of horses in a race ranging from 6 horses to 14 horses, thereby increasing the testing accuracy as a result of less uncertainty due to less horses in a race, while our testing data is only composed of 12-horse races and 14-horse races. A fair comparison of accuracy cannot be done without setting the same number of horses in the testing data for our past model [9] and our current model. In terms of net gain however, our current model can generate a net profit for both 12-horse races and 14-horse races while that of our past model [9] cannot.

In conclusion, comparison with works utilizing neural networks and single horse representations [7] [8] [9] for modeling the races demonstrate that our multiple horse representation is able to achieve comparable win accuracy and superior net gain. This suggests that our multiple horse representation is more suitable than single horse representations for modeling horse racing.

Chapter 9 Conclusion

9.1 Conclusion

This report has detailed the process of using deep probabilistic programming to predict horse racing. Though repeated experiments, we shown that horse racing prediction with deep probabilistic programming and Bayesian neural network can beat public intelligence and generate net profit in the long run under all circumstances, with our best 14-horse model have having accuracy of 22.66% and net gain of 14.43%, and 12-horse model having win accuracy of 22.29% and net gain of 7.54%. We also observed that more neurons per layers are needed for fully capturing the relations when the input dimension is increased, whether it is due to increased number of horses or increases number of features. In addition, our results suggest that both odds data and weather data can be useful for horse racing prediction. Finally, through comparison with related works using single horse representations, our multiple horse representation is able to achieve comparable win accuracy and superior net gain, and is more suitable than single horse representations for modeling horse racing.

9.2 Future Work

One of the main limitations is of our model is that separate models are needed for races with different number of horses. In the future, we plan to research on the direction of transfer learning, network parameter sharing and transformations [1] [2], to alleviate the need of training separate models.

Another short coming of our model is that it is unable to generate accurate confidence. While this does not affect the prediction accuracy of the winning horse, it leads to poor performance when the betting requires an accurate estimate of confidence, such as Kelly betting. Also, although we obtained a distribution from the model, only the mean is used during prediction. In the future, we plan to incorporate the variance of the distribution to obtain an estimate of model confidence for use in betting.

References

[1] F.–F. Li, R. Fergus and P. Perona, "One–shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[2] E. G. Miller, N. E. Matsakis and P. A. Viola, "Learning from one example through shared densities on transforms," in *IEEE Conference on Computer Vision and Pattern Recognition*, Hilton Head Island, USA, 2000.

[3] N. D. Goodman and A. Stuhlmüller, "The Design and Implementation of Probabilistic Programming Languages," 2014. [Online]. Available: http://dippl.org. [Accessed 1 April 2019].

[4] R. N. Bolton and R. G. Chapman, "Searching for positive returns at the track: A multinomial logit model for handicapping horse races," *Management Science*, vol. 32, no. 8, pp. 1040–1060, 1986.

[5] R. G. Chapman, "Still searching for positive returns at the track: Empirical results from 2,000 Hong Kong Races," in *Efficiency of racetrack betting markets*, World Scientific, 2008, pp. 173–181.

[6] W.–C. Chung, C.–Y. Chang and C.–C. Ko, "A SVM–Based committee machine for prediction of Hong Kong horse racing," in *2017 10th International Conference on Ubi–media Computing and Workshops (Ubi–Media)*, Pattaya, Thailand, 2017.

[7] T. T. Cheng and M. H. Lau, "Predicting Horse Racing Result using TensorFlow," Department of Computer Science and Engineering, Hong Kong, 2017.

[8] Y. Liu and Z. Wang, "Predicting Horse Racing Result with Machine Learning," Department of Computer Science and Engineering, Hong Kong, 2018.

[9] Y. Wong, "Horse Racing Prediction using Deep Probabilistic Programming with Python and PyTouch (Uber Pyro)," Department of Computer Science and Engineering, 2018.

[10] A. D. Gordon, T. A. Henzinger, A. V. Nori and S. K. Rajamani, "Probabilistic Programming," in *Proceedings of the on Future of Software Engineering*, Hyderabad, India, 2014.

[11] T. Salismans, D. P. Kingma and M. Welling, "Markov Chain Monte Carlo and Variational Inference: Bridging the Gap," in *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 2015.

[12] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[13] M. D. Hoffman, D. M. Blei, C. Wang and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[14] D. M. Blei and M. I. Jordan, "Variational inference for Dirichlet process mixtures," *Bayesian analysis*, vol. 1, no. 1, pp. 121–143, 2006.

[15] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[16] D. Wingate and T. Weber, "Automated Variational Inference in Probabilistic Programming," *arXiv*, 2013.

[17] W. McCulloch and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[18] S. Haykin, Neural networks, Prentice hall: New York, 1994.

[19] S. Russel and P. Norvig, Artificial Intelligence: A Modern Approach (International Edition), Prentice Hall International, 2012.

[20] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[21] J. Burroni, G. Baudart, L. Mandel, M. Hirzel and A. Shinnar, "Extending Stan for Deep Probabilistic Programming," *arXiv,* 2018.

[22] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall and N. D. Goodman, "Pyro: Deep universal probabilistic programming," *Journal of Machine Learning Research,* vol. 20, no. 1, pp. 973–978, 2019.

[23] R. M. Neal, Bayesian learning for neural networks, Springer Science & Business Media, 2012.

[24] Hong Kong Jockey Club, "Pari–Mutuel Pools," 1 August 2018. [Online]. Available: https://special.hkjc.com/racing/info/en/betting/guide_qualifications_pari.asp. [Accessed 23 November 2018].

[25] Hong Kong Racehorse Owner Association, "Breeding Strategies," 1 August 2018. [Online]. Available: http://www.hkroa.org/en/breeding_strategy.php. [Accessed 23 November 2018].

[26] The Editors of Encyclopaedia Britannica, "Encyclopædia Britannica," Encyclopædia Britannica, inc., 1 August 2018. [Online]. Available: https://www.britannica.com/sports/horse–racing. [Accessed 23 November 2018].

[27] Hong Kong Jockey Club, "Racing 101," 1 August 2017. [Online]. Available: https://entertainment.hkjc.com/entertainment/common/images/learn–racing/racing–academy/pdf/Racing_101_201708_Eng_Final.PDF. [Accessed 23 November 2018].

[28] M. Cox, "Balls and all: why Hong Kong horses are mostly geldings," South China Morning Post, 5 February 2017. [Online]. Available: https://www.scmp.com/sport/racing/article/2068269/balls–and–all–why–hong–kong-horses–are–mostly–geldings. [Accessed 23 November 2018].

[29] Hong Kong Jockey Club, "Racing 201," 1 August 2017. [Online]. Available: https://entertainment.hkjc.com/entertainment/common/images/learn–racing/racing–academy/pdf/Racing201_201708_Eng_Final.PDF. [Accessed 23 November 2018].

[30] S. Han, J. Pool, J. Tran and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," in *Advances in neural information processing systems*, 2015.

[31] S. Han, H. Mao and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *International Conference on Learning Representations*, 2016.

[32] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient–based learning applied to document recognition," *Proceedings of IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutionalneural networks," in *Advances in neural information processing systems*, 2012.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large–scale image recognition," *arXiv*, 2014.

[35] J. L. Kelly, "A new interpretation of information rate," *The Bell System Technical Journal*, vol. 35, no. 4, pp. 917–926, 1956.

[36] D. Tran, "Deep probabilistic programming," *arXiv*, 2017.

[37] Uber Pyro, "Pyro Examples and Tutorials," Uber Pyro, 23 May 2018. [Online]. Available: http://pyro.ai/examples/. [Accessed 23 November 2018].

[38] R. Ranganath, S. Gerrish and D. M. Blei, "Black Box Variational Inference," *arXiv*, 2014.

[39] D. P. Kingma and M. Welling, "Auto–Encoding Variational Bayes," *arXiv*, 2014.

[40] A. Mnih and K. Gregor, "Neural Variational Inference and Learning in Belief Networks," *arXiv*, 2014.

[41] V. Mullachery, A. Khera and A. Husain, "Bayesian Neural Networks," *arXiv*, 2018.

[42] C. Blundell, J. Cornebise, K. Kavukcuoglu and D. Wierstra, "Weight Uncertainty in Neural Networks," *arXiv*, 2015.

[43] S. Levine, "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review," *arXiv*, 2018.

[44] J. M. Hernández–Lobato, M. A. Gelbart, R. P. Adams, M. W. Hoffman and Z. Ghahramani, "A General Framework for Constrained Bayesian Optimization using Information–based Search," *arXiv*, 2015.

[45] A. Shah and Z. Ghahramani, "Parallel Predictive Entropy Search for Batch Global Optimization of Expensive Objective Functions," *arXiv*, 2015.

[46] J. M. Hernández–Lobato, M. W. Hoffman and Z. Ghahramani, "Predictive Entropy Search for Efficient Global Optimization of Black–box Functions," *arXiv*, 2014.

[47] S. Thrun, W. Burgard and D. Fox, Probabilistic Robotics, MIT Press, 2005.

[48] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[49] K. P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.