# Trot Racing Winning Probability Prediction Notebook

A Regression Neural Network Model with Deep-Learning Algorithms

Fangzheng Zhang, Ruining Liu, Xiaodong Lu

## I. Pre-process, Variables, Cleaning Assumptions

French trot racing involves a lot of information about the contestant. In this project, we aim to predict the probability of winning for each observation. That is, for each instance of a particular horse participating in a specific competition.

We begin with data cleaning and filtering out the variables for later use. First, as instructed, we separate our training set from the hold-out set by the date before and after Nov. 1st, 2021, respectively. Then, after assessing the relevance of the variables with the winning result, we choose 10 variables as predictors for winning probability. The 10 variables are: Barrier, Distance, Gender, HandicapDistance, Hindshoes, HorseAge, RacingSubType, StartType, StartingLine, and Surface. We set the response variable to be FinishPosition.

For training purposes, we filter out the observations that had empty values in these variables, as well as the ones that had FinishPosition that does not have an integer entry, which included “BS” (Break Stride), “DQ” (Disqualified), “FL” (Fell), “NP” (Took no Part), “PU” (Pulled Up), “UN”, “UR”, and “WC”. We decide not to consider these terms in the final outcome of the rank, which is numerical.

The cleaning process ensures that our training set has only observations with accessible, valid values for further operations.

## II. Model

Our model follows the regression model in statistics, incorporating ideas from machine learning, such as the concept of neural network.

We rely on a PyTorch implementation which demonstrates a regression model for horse racing predictions.

RegressionModel class definition:

We use PyTorch as our architecture to create a neural network model for regression. It inherits from nn.Module, which is the base class for all PyTorch models.

The constructor (__init__ method) defines a feedforward neural network with four linear (fully connected) layers. Each torch.nn.Linear layer specifies the input size and output size for the layer.

Here, the forward method defines the forward pass of the neural network, specifying the order in which data flows through the layers with ReLU activation functions. By using ReLU activation function, we have introduced more nonlinearity into the model.

Instantiating the model:
The code instantiates the RegressionModel by providing the input_size, which is determined by the number of feature columns in the dataset.

Loss function and optimizer:
We choose to use Mean Squared Error (MSE) loss (nn.MSELoss) as the loss function for regression tasks. The Adam optimizer (optim.Adam) is chosen to optimize the model's parameters, with a learning rate of 0.001.

Preparing data loaders:
We specify the batch size, which is 64, and the number of training epochs, which is 30. After the evaluation, we choose the parameters which make the most accurate predictions, which is horse_racing_model.pth. It has the smallest Deviation in our evaluation model, eval_model.py, which will be explained in the next part.

We create a custom dataset by selecting variables we decide to use for training our model. We load the data from the CSV file. Those variables are the features used to train the model, and we choose FinalPosition as our label. Then, we create the instance of HorseRacingDataset, input it into DataLoader to handle batching and shuffling of the data.

Training the model:
The code enters a loop over epochs and iterates through the data in batches.
For each batch, the model performs the following steps:

1. Forward pass: Passes the batch of input features through the model to obtain predictions. Calculates the mean squared error (MSE) loss between the predictions and the actual labels.
2. Backward pass: Computes gradients of the loss considering the model's parameters.
3. Optimizer step: Updates the model's parameters using the optimizer.
4. Periodically prints the loss to monitor training progress.
5. After each epoch, it saves the model's state dictionary to a file.

**The final predicted probability is stored in the last column "Probability".**

## III. Testing

To test the effectiveness of our prediction model, we created an evaluation model. We used the idea of residual from statistics, where we measure the deviation from the predicted value and the actual value. Here, we are using the values from the variable "FinishingPosition" from the hold-out set.

The total deviation is calculated by:

$$\text{Deviation} = \sum_{n=1}^{n} \frac{(predicted - actual)}{m}$$

where n is the total number of competitions, and m is the number of contestants at each race.

The codes are stored in the document eval_model.py.