# If (A)I were a Betting Man: Profitable Sports Betting with Deep Reinforcement Learning

Stanford CS 229 Final Project – Theory & Reinforcement Learning Category

**Anthony Weng**
Department of Computer Science
Stanford University
ad2weng@stanford.edu

## Abstract

With sports betting being a popular and potentially lucrative pastime, professional bettors often develop complex systems employing private, sport-specific knowledge in an attempt to make profitable bets. We explore the efficacy of using a deep reinforcement learning approach to learn a profitable betting policy using a freely available, sport-agnostic source of information: an outcome's moneyline itself. We compare this approach's performance against multiple baselines, including the policy embodied by a logistic regression model and the heuristic strategy of betting the favorite only. The reinforcement learning agent demonstrates improvements over the logistic regression policy, but falls short of the performance of the favorites-only strategy in multiple desirable criterion (e.g., profitable bet rate, overall profit, etc.). We introspect on the workings of the deep reinforcement learning system and provide commentary on potential solutions as well as future work directions.

**Key information:**

Mentor: Garrett Thomas – no external collaborators – no project sharing.

## 1 Introduction

Sports betting has long been a favored pastime for sport spectators, capable of increasing their perceived level of engagement with a given sporting event by creating a personal financial stake in its outcome. Digital technologies like sportsbook apps (e.g., *FanDuel*, *DraftKings*) have democratized sportsbook access and simplified the betting process – and with the bettor influx necessarily comes those who aim to make a profession out of beating the odds. These dedicated odds analysts employ expert prediction models, arbitrage strategies, and other complex systems in attempt to regularly place profitable bets [1].

While these approaches are capable of generating consistent, long-term returns, their execution can rely on sport-specific knowledge, proprietary assessments, and/or insider information which further obscures the anatomy of such methods. In response to these guarded systems, we aim to develop a profitable betting model using a source of information available to all – an outcome's moneyline itself – by training a betting agent via deep reinforcement learning. In doing so, we hope to find that machine learning technologies can convert even imperfect, public signals of the wisdom of the crowd (i.e., the moneyline history) into a long-term, profitable strategy.

## 2 Related Work

Deep reinforcement learning (DRL) is a sub-field of reinforcement learning (RL) which combines the representational power of neural architectures with the agent-environment learning structure which underpins traditional RL. DRL has been applied to various domains to great effect, including games, robotics, natural language processing, finance, etc. [2]. Here, we seek to extend DRL's successful reach to include sports betting. This addition would follow as a natural convergence of independent yet related machine and reinforcement learning works. For instance, Cornman et al. explored professional tennis match betting as a domain for non-DRL machine learning [3], while Dahl trained RL agents to learn optimal betting strategies for the common poker variant, Texas Hold'em [4]. Betting markets have already been shown to be exploitable by intelligent strategies [1], and when using neural architectures, researchers have found that models can predict event outcomes surprisingly well with only the information embedded in public betting market figures [5]. Following these early signals of profitability, others have already

begun to fine-tune RL-for-betting methodologies to induce further gains (e.g., Walsh et al. optimizing machine learning betting models for calibration instead of accuracy [6]). Collectively, this body of prior literature suggests that a DRL approach using only publicly-available information (such as the moneyline history) has the potential to achieve long-term profitability – which is precisely the premise we seek to explore in this work.

# 3 Approach

## 3.1 RL environment definition

We train a betting agent via deep reinforcement learning using a three-layer multilayer perceptron (MLP) to define our agent's decision-making neural architecture. Key elements of the reinforcement learning environment (RLE) experience tuple $(s_t, a_t, r_t, s_{t+1}, d_t)$ are defined as follows:

- State ($s_t$): a vector $\mathbf{v} \in \mathbb{R}^{145}$ which contains: (1) the concatenated 72-hour histories of by-hour moneylines for the home and away teams; and (2) the agent's current total balance. Initial balance amounts for agent training and evaluation are discussed in **Sections 3.2** and **3.3**.
  - State transition mechanics ($s_{t+1}$): In practice, the actions and payout received by a single bettor have minimal influence on the betting odds presented for other (potentially unrelated) events in a following timestep. To better reflect this fact, we construct the state transition to be random; i.e., the home and away moneyline histories in timestep $t + 1$ are randomly sampled from the dataset and are not influenced by those of timestep $t$. Note that the agent balance is updated deterministically.
- Action ($a_t$): a 0 to 1 probability. Probabilities in the range [0, 0.5) correspond to betting on the home team whereas those in the range [0.5, 1] correspond to betting on the away team. The distance from the decision boundary probability (i.e., 0.5) determines what proportion of a "standard moneyline bet" (defined in **Section 3.2**) is placed on the selected team. Probabilities further from 0.5 denote a bet of greater magnitude. For example:
  - if probability $= 0$, the agent will place a *full* standard moneyline bet on the home team; and
  - if probability $= 0.25$, the agent will place $\frac{|0.25-0.5|}{0.5} =$ *half* of a standard moneyline bet on the home team.
- Reward ($r_t$): the payout from betting the proportional amount of a standard moneyline bet (as determined by $a_t$) on the selected team given the outcome of the match. After each timestep, the agent balance is updated as: balance $:=$ balance $+ r_t$.
- State terminality ($d_t$): states in which the reward amount results in a loss such that the agent's balance $\leq 0$ are regarded as terminal states; all other states are non-terminal.

## 3.2 DRL methodology and baselines

Given the technically infinite number of possible 72-hour moneyline histories, we elect to use a model-free algorithm to train our agent. Specifically, we employ deep Q-learning to circumvent the need to design quality exploration policies (which we lack the requisite expert knowledge to do so). Also, two key RL/DRL training techniques are employed: (1) an $\epsilon$-greedy action selection regime; and (2) use of training and target networks. These techniques help to limit early convergence to locally optimal solutions and to encourage stability in the network's Q-value estimations, respectively. **Section 3.3** provides additional model configuration details (network layer sizes, weight initializations, etc.).

We design and implement three baseline strategies to compare the DRL-trained agent against:

1. The policy represented by a logistic regression model, demonstrating how much traction a simpler ML technique can gain on the problem domain prior to introducing more sophisticated, compute-intensive model architectures.
2. The heuristic strategy of always betting the favorite, helping to separate DRL model success from dataset artifacts related to how accurate the underlying odds data are.
3. The strategy of randomly betting, enabling comparisons against a minimal baseline to ensure we are making meaningful traction on the problem in an absolute sense.

# 4 Experiments

## 4.1 Data

The data employed for this work are a subset of those collected by Kaunitz et al. [1] in their associated work. The original dataset includes 14 months' worth of per-match time series odds, with each time series

reporting hourly sampled odds of winning for both the home and away teams from up to 32 bookmakers for 72 hours till the start of each game. The games in question are English football matches (also referred to as soccer).

For this work, we construct an simplified, aggregated dataset $\mathcal{D}$ as follows:

1. Subset the original dataset to entries corresponding to matches played in the 2015 calendar year;
2. Filter entries which concluded in a draw;
3. For both the home and away teams, consider the *median* odds figure presented by the up to 32 bookmakers to be the representative odds of that team winning;
4. Remove entries which do not possess complete, 72-hour odds histories for both the home and away teams; and
5. Convert aggregated odds figures to moneyline figures.

We apply the no-draws filter to enable problem framing as a binary classification task (i.e., an output of $0$ can correspond to the home team winning, and a $1$ the away team winning). The revised dataset $\mathcal{D}$ contains 9994 aggregated entries and is randomly shuffled and divided into training, validation, and test splits each containing 60/30/10 percent of $\mathcal{D}$, respectively.

### 4.2 Evaluation methods

To evaluate a given model or strategy, we initialize a decision-making agent with a starting balance of \$1000. The agent sequentially places bets on the moneyline histories (i.e., states in our RL environment) according to the model or strategy being executed. After a bet is placed, the payout (given the outcome of the corresponding match) of said bet is computed and used to update the agent's balance according to the following rule: balance $:=$ balance $+$ payout. Evaluation continues in this way until all test set states are processed (of which there are approximately 1000) and/or the current agent possesses a non-positive balance after the most recent bet and associated balance update.

To ensure fairness with respect to the test set ordering, all agents process test set states in the same order. All bets are evaluated using the final (i.e., hour 72) moneyline figure offered for the bet-upon team. The specific amount an agent bets is determined by whether a given agent is using a heuristic strategy or model-represented policy. Agents who use a heuristic strategy can only generate a 0/1 output which corresponds to placing a *full* "standard moneyline bet" on the home or away team, respectively. We define a standard moneyline bet as follows:

- If the final moneyline $m$ offered for the bet-upon team is positive (i.e., $+400$), our model/strategy agent will bet \$100, resulting in a net earning of $\$m$ if the bet-upon team wins (i.e., the bet "hits") and a net loss of \$100 otherwise.
- If the final moneyline $m$ is negative (i.e., $-2000$), our model/strategy agent will bet $\$m$, resulting in a net earning of \$100 if the bet-upon team wins and a net loss of $\$m$ otherwise.

As described in **Section 3**, agents who execute model-represented policies (i.e., policies learned through logistic regression or DRL) generate 0 to 1 probabilistic outputs for which values in the range [0, 0.5) correspond to betting on the home team and those in the range [0.5, 1] to betting on the away team. Values further from 0.5 indicate more certainty in the relevant team winning and therefore are reflected in a bet of larger magnitude. Specifically, depending on the exact value of the probability $p$, a multiplier $m$ is applied to the value of a standard moneyline bet to compute the agent's *proportional* bet. $m$ is computed as $m = \frac{|p-0.5|}{0.5}$ for all values of $p$.

For each agent and their respective model or strategy, various metrics for the test set matches including (but not limited to) profitable bet rate, overall profit, and average amounts won and loss are computed (see Table 2). We also compute hit rate and average winnings for three types of matches of interest (see Table 3), which are defined as follows:

1. *Underdog upsets*[1] in which the underdog (as determined by the moneyline figures) wins the match.
2. *Closely contested* matches in which the moneylines for both teams are within 400 of each other.
3. *Clear favorite to win* matches in which one team's moneyline is negative while the other's is positive, indicating that the odd-makers strongly believe the team with the negative moneyline should win.

[1] In betting terminology, the term "underdog" refers to the team/player less favored to win the game as reflected by a less negative/more positive moneyline.

### 4.3 Experimental details

#### 4.3.1 Neural architecture specification and training

As previously noted in **Section 3**, we employ a deep Q-learning approach to train our RL agent. A three-layer MLP is used to represent the agent's Q-function, with the input, hidden, and output layers having 145, 145, and 2 nodes, respectively. Our network uses the ReLU activation function, with network parameters being initialized via Xavier uniform initialization.

Network loss for a single training set experience is computed as follows:

$$\text{Loss}(s_t, a_t) = \left[Q(s_t, a_t) - (r_t + (1 - d_t) \cdot \gamma \max_{a'} Q(s_{t+1}, a')\right]^2$$

which is reflected in our use of a mean squared error loss function. We update network parameters via stochastic gradient descent with a batch size of 1. Akin to test set evaluation, our neural agent is assigned a initial balance for each training episode. Training episodes end when either (1) the neural agent has experienced all states represented in the training set; and/or (2) the neural agent's balance for that episode reaches a non-positive value (with balance updates being conducted in the same fashion as described for agent evaluation in **Section 4.2**). The initial balance assigned is \$6000 which is proportional to the balance assigned during agent evaluation given the relative sizes of training and test data splits.

#### 4.3.2 Hyperparameter values

54 hyperparameter configurations were tested via grid search with manually-defined search ranges. The final hyperparameter configuration used for test set evaluation is that which obtained the highest overall profit on the validation data split (using the evaluation protocol described in **Section 4.2** with an initial balance of \$3000 – again, proportional to the \$1000 used for the test set evaluations given the relative sizes of the validation and test data splits).

Table 1: Utilized and searched hyperparameter values.

| Hyperparameter | Description | Search Range | Final Value |
|---|---|---|---|
| $\epsilon$ | Starting probability for $\epsilon$-greedy action selection. | [0.3, 0.5, 0.7] | 0.3 |
| $\epsilon$ decay factor | Amount to decay $\epsilon$ per training episode (min. $\epsilon$ = 0.05). | NA | 0.001 |
| $\gamma$ | RL environment discount factor. | [0.95, 0.99] | 0.95 |
| `num_train_eps` | Number of training episodes. | NA | 1000 |
| `num_exps_to_copy` | # experiences per training and target network weights sync. | [10K, 20K, 50K] | 20K |
| $\eta$ | SGD optimizer learning rate. | [1e-2, 1e-3, 1e-4] | 1e-3 |

### 4.4 Results

Table 2: Basic test set metrics for evaluated strategies.

| Strategy | Test set bets made | Profitable bet rate (%), # profitable bets made | Overall profit (\$) | Avg. amount won (\$) | Avg. amount lost (\$) | Largest amount won (\$) |
|---|---|---|---|---|---|---|
| Favorite-only | 1000 | 68.1 (681) | 43,040.00 | 110.03 | 100.00 | 170.00 |
| Random | 125 | 55.2 (69) | -7,980.00 | 166.96 | 348.21 | 750.00 |
| Logistic Regression | 1000 | 65.4 (654) | 17,652.90 | 40.44 | 25.41 | 108.24 |
| Deep Q-learning | 1000 | 58.6 (586) | 24,592.08 | 126.91 | 120.24 | 660.70 |

Table 3: Test set metrics for match types of interest by strategy.

| | Underdog upset | | Closely contested | | Clear favorite to win | |
|---|---|---|---|---|---|---|
| Strategy | Hit rate (%) | Avg. winnings (\$) | Hit rate (%) | Avg. winnings (\$) | Hit rate (%) | Avg. winnings (\$) |
| Favorite-only | NA | NA | 60 | 119.85 | 77 | 100.00 |
| Random | 44 | 257.41 | 57 | 156.25 | 57 | 177.50 |
| Logistic Regression | 36 | 14.44 | 56 | 17.81 | 76 | 59.57 |
| Deep Q-learning | 34 | 230.27 | 51 | 134.42 | 68 | 119.55 |

Inspecting Table 2, we observe that the favorite-only heuristic strategy outperforms all other strategies in terms of both profitable bet rate and overall profit. Note that the highly positive profit garnered by betting the favorite only is almost certainly a data construction artifact; i.e., the matches which these moneylines represent are soccer games which routinely end in ties. Our deep Q-learning agent did collect a larger total profit than the logistic regression agent, despite the former achieving a lower profitable bet rate – offering evidence that more sophisticated ML techniques can provide addition traction for optimal betting problems. This result may also suggest that a DRL approach can train a betting agent which is not necessarily correct more often than its peers, but when the DRL agent is correct, it is "more correct" or more correctly certain (note how the DRL agent achieves the second-highest amount won on any bet).

When examining performance on match types of interest (see Table 3), the DRL agent fails in a rather spectacular way. Across all types of matches of interest, it bets on the winners at lower rates than the logistic regression agent. The same can almost be said of the DRL agent and the random agent, save for the fact that the DRL agent achieves a higher hit rate when the odds present a clear favorite to win (which is likely the least difficult type of match to bet upon). We do note that the DRL agent achieves some of the highest average winnings per hit (almost comparable to the figures of the random agent), which provides further evidence for the previous suggestion that our DRL approach is not correct more often but more certain when it is correct as compared to the other strategies.

## 5 Discussion

In theory, the favorite-only strategy should represent a floor on the performance achievable by a DRL betting agent. Reason being (assuming the test and training data splits are similarly distributed), a DRL agent can always learn a policy which embodies the favorite-only strategy. Concretely, this could have been accomplished in our network by the agent learning to ignore all inputs besides non-final odds figures and simply betting on the team which corresponds to the more negative/less positive number. Moreover, since the DRL agent is also capable of modulating the proportion of a standard moneyline bet made, it should be capable of learning to minimize the potential (betting) losses for events of which the outcome is relatively uncertain given the final odds figures for each team.

Then, we must introspect on why this result was not observed in practice. When inspecting the DRL agent outputs for the test set, we found that the DRL agent had adopted the degenerate policy of always betting on the home team. This policy may have been learned as a result of multiple factors:

1. The decision-making neural architecture may have been too simple and/or too unsophisticated. With a low number of training episodes and a small training set, the relatively shallow and simplistic architecture we employed may have both lacked sufficient exposure to and possessed limited ability to learn patterns in the training data (like match favorites often winning).
2. The loss formulation employed here may have not been appropriate and thus not conducive to learning a sophisticated policy. For bet computation, we normalize the network's Q-value estimations for betting on the home/away teams and select the higher value as the agent's team to bet upon. This process essentially produces a probability distribution over the two teams winning and thus may be more suited to use of a cross-entropy loss function.
3. We observe that the training process was highly fragile with respect to hyperparameter choices. Hyperparameter configurations which only differed in one aspect often converged at vastly different speeds, with many configurations resulting in extremely fast convergence. This suggests our hyperparameter search likely needs to be conducted in a more sophisticated manner to ensure training stability and prevent learning of a degenerate policy.

Implementing changes with respect to these issues may be critical to DRL approaches gaining the most traction possible on betting-related environments. If after doing so a more sophisticated policy can be represented and learned, fine-tuning aspects of the RL setup could then lead to further profitability gains (e.g., altering the reward function to penalize errors on highly profitable match types, like underdog upsets and closely contested matches, more so than on others).

## 6 Conclusion

Sports betting is an emerging field for the application of machine and reinforcement learning techniques. Here, in a union of previous approaches, we apply a deep reinforcement learning methodology to train an agent to bet on the winner of soccer matches using only publicly disclosed information – namely, each participating team's moneyline history for the 72 hours preceding the match. When comparing the performance of the trained agent to that of betting the favorite only (a strategy which intuitively should net near-zero profit in real-world applications), the DRL-trained agent fails to outperform the favorite-only strategy. While perhaps disappointing, we do note the DRL-trained agent outperformed a logistic regression-trained agent in terms of overall profit, which suggests that a more carefully engineered DRL methodology could gain additional traction on the problem.

Supposing a more robust DRL methodology, future researchers in this space could extend the work done here in numerous ways. For instance, adapting the DRL agent to operate in settings where there are more than two potential event outcomes (e.g., win, tie, draw) could increase such a work's real-world applicability. A DRL agent could also be trained to bet on several event outcomes simultaneously, creating a bet portfolio that allows for the management and minimization of overall portfolio risk. One could also experiment with providing the DRL agent with still public but event-specific additional information (e.g., player or team statistics) in an attempt to further improve event-specific profitability. In short, there is a wealth of potential developments for this DRL application, and we look forward to seeing what others do.

## Contributions Statement

All project work was completed by the sole author.

## References


[1] Lisandro Kaunitz, Shenjun Zhong, and Javier Kreiner. Beating the bookies with their own numbers - and how the online sports betting market is rigged, 2017.

[2] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[3] Andre Cornman, Grant Spellman, and Daniel Wright. Machine learning for professional tennis match prediction and betting. *Stanford Unverisity*, 1(2):4, 2017.

[4] Fredrik A Dahl. A reinforcement learning algorithm applied to simplified two-player texas hold'em poker. In *Machine Learning: ECML 2001: 12th European Conference on Machine Learning Freiburg, Germany, September 5–7, 2001 Proceedings 12*, pages 85–96. Springer, 2001.

[5] Sascha Wilkens. Sports prediction and betting models in the machine learning age: The case of tennis. *Journal of Sports Analytics*, 7(2):99–117, 2021.

[6] Conor Walsh and Alok Joshi. Machine learning for sports betting: should forecasting models be optimised for accuracy or calibration? *arXiv preprint arXiv:2303.06021*, 2023.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.