Predicting Formula One Race Results: A Deep Learning Approach

CS229 Final Project

Isaac Kleisle-Murphy

Abstract

In this paper, I examine how three ubiquitous machine learning algorithms – namely, multinomial logistic regression, feed-forward neural networks, and convolutional neural networks – can be leveraged to predict the outcomes of modern Formula One races. Under my pipeline data processing and feature engineering pipeline, I ultimately show that the feed-forward neural network performs the best of these three, achieving a cross entropy of 1.909 when tested on the 2019 and 2020 Formula One seasons.

Code

`https://github.com/isaackleislemurphy/formula-one`

Stanford University
Department of Computer Science
11/18/2020

1 Introduction and Project Motivation

Formula One ("F1") is an international single-seater auto racing series widely regarded as the pinnacle of motor sport. On approximately twenty weekends each year ("Grands Prix"), the world's top racing drivers compete at various venues across the globe. Driving ultra-aerodynamic, state-of-the-art open-wheel race cars, these drivers navigate winding race circuits inches apart from one another – sometimes in excess of 240 mph and at six times the force of gravity.

Given its technical spirit, strategic complexities, and data-driven ethos, F1 is ripe for machine-learning applications. Teams themselves fervently engage in such pursuits, though given the hyper-competitiveness of the sport, their proprietary models remain internal. Yet curiously, in the public arena, there exists surprisingly few predictive analyses, in sharp contrast to other data-driven sports, such as baseball, ice hockey, or basketball. Accordingly, I propose one such application – namely, a feed-forward neural network that, at the outset of a race, predicts a driver's finishing position, given both the circuit's competitive characteristics as well as the drivers' performances (qualifying times, previous race results, maximum speeds, etc.) leading up to the race.

2 Literature Review

As set forth above, there exists surprisingly little statistical and/or machine-learning literature analyzing F1. The most prominent F1-adjacent analyses tend to focus on driver rankings across history. Most visibly, FiveThirtyEight, Amazon Web Services, and the popular "F1 Metrics" blog have all published historical driver rankings, to the mixed reception of fans and pundits. Elsewhere, Amazon Web Services provides in-race models/graphics relating to car performance, tyre degradation, and overtake probabilities during F1 broadcasts; however, these graphics have garnered reputation more for their ability to strain credulity than their ability to make in-race predictions. As it relates to actual race predictions, the website "f1-predictor" distributes weekly race predictions via an unknown methodology.

3 Data Collection, Preprocessing, and Feature Engineering

My principal data source was the Ergast F1 API, which provides basic timing and box-score statistics from each F1 race between 2006 and 2020.[1] For my purposes, relevant variables provided by the API included: the start and finishing positions of each driver in each race; the lap times of each lap completed by each driver in each race; the best qualifying session times of each driver prior to each race; and the fastest speed achieved by each driver in each race.

Importantly, this data is only a sliver of that which is available internally to F1 teams. Whereas this public data contains simple lap times, qualifying times, fastest lap speeds, pit stop times, and box score-esque results, teams have access to, among other variables: sector times, GPS telemetry, aerodynamic sensor measurements, and much more. As such, the model presented here will not be competitive with those internal models. Nevertheless, my aim is to maximize that which is publicly available and reasonably accessible.

The preprocessing and feature engineering pipeline for this model was extensive. Before being fed to the model, the raw Ergast data endured a lengthy cleaning, wrangling, and imputing process. While a precise description can be found in 'processing.py', the following paragraphs provide an overview of how the data was ingested, wrangled, and organized. Note that before entering any model, features were scaled via min-max scaling.

3.1 Performance Differential Features

Given that a.) qualifying time[2] is generally understood as a benchmark for overall driver/car speed, and b.) qualifying results have a deterministic effect on the race's results, I devoted considerable attention to the inclusion of qualifying-specific features. First, for each Grand Prix, I took each driver's fastest qualifying lap and computed the inter-driver time differences.[3] These inter-driver deltas (measured in seconds) were then assembled into a 24×24[4] matrix and ordered by starting grid position. In other words the i^{th} row and j^{th} column of this matrix contained the fastest qualifying time of the driver starting in the i^{th} grid position, minus the fastest qualifying time of the driver starting in the j^{th} grid position. As a single example, consider the qualifying delta matrix from the 2020 Russian Grand Prix, as visualized in Figure 1:

Here, each entry reflects an ordered pair of starting grid positions, and the color reflects the qualifying time delta between said pair (delta = row - column). As one might expect, the deltas between the back of the grid[5] and the front are understandably greater in magnitude, with the slowest qualifying drivers/cars setting much longer times than the fastest qualifying drivers/cars; conversely, the deltas between adjacent grid positions are much smaller, with less time separation between these more closely-matched drivers/cars. In the example in Figure 1, the P15 row stands out, with much smaller qualifying deltas

[1] There are approximately 18-21 Grands Prix per season.

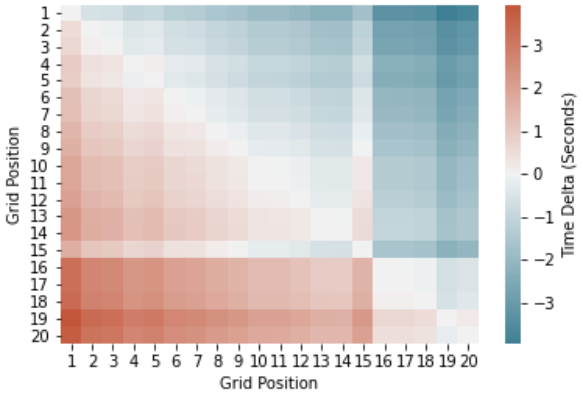
[2] "Qualifying" describes the time trial competition scheme used to determine the starting order of the race.

[3] Only the fastest lap times of each race's qualifying sessions was reported in the data; relatedly, more recent races involve three "knockout-style" qualifying sessions, whereas mid-2000's races involved "one-shot" qualifying.

[4] 24 was chosen because the maximum number of drivers in any race between 2006 and 2020 was 24; races with less than 24 drivers had corresponding cells filled with zeros, with ones (to signal the missingness of these drivers).

[5] Formula One positions are often referenced as Pi, short for "Position i".

Figure 1: Qualifying Delta Matrices, 2020 Russian GP



to the front of the grid than to the more proximate grid positions. This occurred because the driver in this race, Alex Albon, originally qualified P10. However, he was assessed a five-place grid penalty, and thus he started the behind his original qualifying position (i.e. below his true pace). The matrix captures this nicely, indicating that Albon/Albon's car may be faster than his P15 grid position would indicate.

Similar to the qualifying delta matrix above, I also constructed analogous matrices for qualifying differentials and top speed differentials, albeit averaged over the previous races in a particular season.[6] In so doing, I hoped to capture seasonal performance deltas between teams/drivers, in order to more fully communicate the competitive hierarchy of a particular race. This way, the model could learn not just from a single race's qualifying session (where some drivers may have performed above or below their seasonal tendencies), but also from less mercurial seasonal performance trends. Through these deltas, I was providing multiple "images" of the field's competitive spread – with respect to both a particular qualifying session and the overall season – in an efficient and compact format.

3.2 Previous-Race & Box Score Features

To supplement the driver performance deltas described above, I also engineered basic measures of driver and car performance, including: driver/team win rate, driver/team podium rate, driver/team top ten rate, driver crash rate, and team mechanical failure rate, both in the current season and as a rolling three season average. I further included the race number, to contextualize the in-season rates[7], as well as each driver's number of career starts, to capture driver experience and "savvy."

3.3 Circuit Features

The venue of a Grand Prix can also impact race outcome. For example, some tracks (e.g. Monaco) are narrower, precluding overtaking, while others (e.g. Spa-Francorchamps) induce more accidents. To account for these considerations, I engineered the following track-specific features: Pearson correlation between start and finish position (to capture ease of overtaking) over the past ten seasons; accident rate over the past ten seasons (to capture attrition); the number of times each driver had previously raced at the circuit in their career (to capture driver familiarity with the circuit); a track-specific indicator for all tracks to have hosted three or more Grands Prix in the training data window (to capture intrinsic track characteristics); and an indicator of whether or not a race was run in the wet or dry.

4 Modeling

4.1 Overview

As set forth above, the model is designed to output the probability vector of each of the top ten finishing positions, or eleventh or worse. That is, for each race $i = 1, \ldots, N$, starting grid position $j = 1, \ldots, J$, corresponding feature vector $x^{(i,j)}$, and corresponding finishing position $F_{i,j} \in \{1, 2, \ldots, 10, 11^+\}$, I predict softmax vector

$$P(\vec{F_{i,j}}|x^{(i,1)}, \ldots, x^{(i,j)}) = \langle P(F_{i,j} = 1|x^{(i,1)}, \ldots, x^{(i,j)}), \ldots, P(F_{i,j} = 11^+|x^{(i,1)}, \ldots, x^{(i,j)})\rangle.$$

That is, for each driver in each race, I sought to generate a vector containing the probabilities of finishing in positions $k = 1, 2, \ldots, 10, 11^+$. I then fit four basic models, to wit,

1. *A naive model, which simply returns the historical finishing probabilities of drivers starting the race from a particular starting grid position.* For example, if a driver were starting a race from P3, this naive model would return a vector consisting of: the proportion of wins ($k = 1$) among drivers starting P3 ($j = 3$) in previous races; the proportion of P2 finishes ($k = 2$) among drivers starting P3 in previous races; ...; and the proportion of P11$^+$ finishes ($k = 11^+$) among

[6] Unfortunately, top speeds from qualifying sessions were not included in this data.
[7] A crash rate of 1 entering the second Grand Prix is much different than a crash rate of 1 entering the eighth Grand Prix.

drivers starting P3 in previous races. Formally, for race $n+1 \leq N$ and each starting position $j = 1, \ldots, J$, this model returns the sample probability vector over all of the n previous races:

$$\frac{1}{n}\left\langle \sum_{i=1}^{n} \delta(F_{i,j} = 1), \ldots, \sum_{i=1}^{n} \delta(F_{i,j} = 11^{+}) \right\rangle.$$

2. *A multinomial logistic regression ("MLR"), with L2 regularization (ridge) and no interaction terms.* In this model, each "row," or feature vector, is a flattened vector of the aforementioned features corresponding to a particular race. Importantly, for each of the three performance differential matrices, the matrix row corresponding to the driver's starting position is extracted and appended to the feature vector for this race. Formally, the model is

$$P(F_{i,j} = k) = \frac{\exp(\beta_k^T x^{(i,j)})}{\sum_{\ell=1}^{K} \exp(\beta_\ell^T x^{(i,j)})} \quad \text{subject to} \quad \arg\max_{\beta} \sum_{i=1}^{N} \sum_{j=1}^{J} \log \prod_{k=1}^{K} \frac{\exp(\beta_k^T x^{(i,j)})}{\sum_{\ell=1}^{K} \exp(\beta_k^T x^{(i,j)})} + \lambda \sum_{k=1}^{K} ||\beta_k||_2^2,$$

where again, $F_{i,j} \in \{k = 1, 2, 3, \ldots, 10, 11^{+}\}$ describes the finishing position of grid position j in race i.

3. *A feed-forward neural network ("FFNN"), with an identical input/output architecture to that of the multinomial logistic regression.* More formally, this model consists of a directed structure of neurons, weights, and activation function. Within a fully-connected feed-forward network, each node (part of a layer of neurons) reflects a weighted combination of neurons in the previous layer, followed by an application of a $\mathbb{R} \to \mathbb{R}$ activation function. That is, for layers $r = 1, \ldots, R$, the vector of neurons for the r^{th} layer is given by

$$a^{[r]} = g(W^{[r]} a^{[r-1]} + b^{[r]}),$$

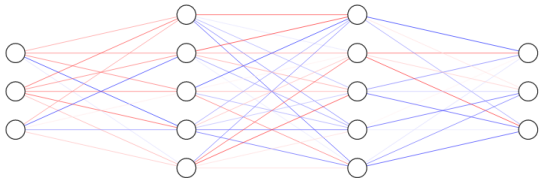
where $W^{[r]}$ is a matrix of weights for this r^{th} layer, $b^{[r]}$ is a vector of biases for this r^{th} layer, and $g : \mathbb{R} \to \mathbb{R}$ activation function applied to each element in the linear combination. Importantly, randomized "dropout" – i.e. the randomized deletion of neurons from the network at some tunable rate – is employed here as a means of regularization.

4. *A convolutional neural network ("CNN") with two input streams: (1) an image and (2) a vector of additional features.* The first stream, the "image," consists of the three 2D performance differential matrices, as stacked into three channels. These "image" inputs undergo two layers of 2-dimensional convolutions and max-pooling, before being flattened and merged with the vector of additional features into a traditional feed-forward structure. Formally, the convolutional component of model works as follows: building on the sequential weight/bias/activation detailed above, a kernel matrix scans the multidimensional input image/matrix and extracts features relevant to prediction. A max-pooling layer then extracts the most prominent attributes/"findings" from this convolutional layer. The output of this max-pooled layer can then be passed to further convolutional layers and feature extraction, before eventually being flattened and/or merged into a forward-connected structure.

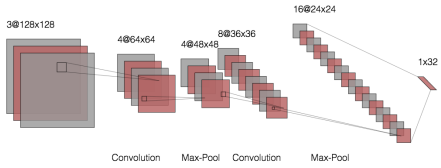
   Importantly, the structure of this model is distinct from that of the previous two. Whereas the MLR and FFNN ingest a vector of features corresponding to a single driver/race pair ($< j, i >$), and output a vector of finishing probabilities for that driver only, this model takes in a matrix of features for all $j = 1, \ldots, J$ drivers in race i and returns J probability vectors – one for each driver[8] – which are the $P(\vec{F_{i,j}})$. I believed this approach warranted consideration, given the CNN's well-known ability to learn spatial relationships. I hypothesized that providing the model with a 2D matrix of performance deltas – in effect, an "image" of the field's competitive spread – might dovetail nicely with the spatial learning capabilities of the CNN.

Figure 2: Field Spread Feature Matrices (2020 Russian GP)

(a) Example of a Feed-Forward Neural Network



(b) Example of Convolutional Layering



[8]That is, whereas models 2 and 3 are single-output, multi-class models, this model is a multi-output, multi-class model.

4.2 Model Selection

Races were split into a training and set (seasons 2006-2018; 246 Grands Prix) and a test set (seasons 2019-2020; 31 Grands Prix). Within the training set, each of models were tuned via a single repeat, five-fold, randomized cross validation procedure over a grid of standard hyperparameters[9]. Cross entropy,[10] as averaged over the five validation folds, was the principal validation/tuning criteria. Importantly, each of the five folds was identical over each of the models and tune settings, in order to ensure direct comparison between hyperparameter settings and/or models. The optimal tuning settings, as found via this five-fold CV grid search, are displayed below:

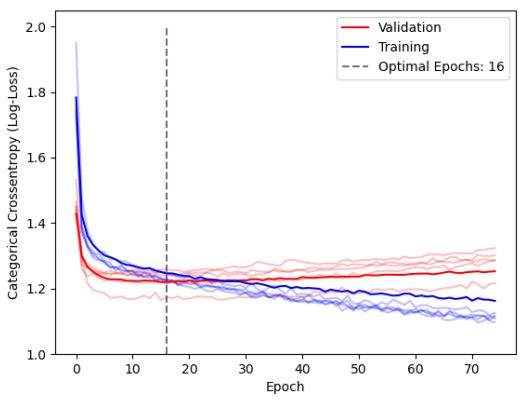
Table 1: Tuning/Validation Results

Model	**Avg. Val Cross Entropy**	**Corresponding Hyperparameters**
MLR	*1.2536*	$C = 1/\lambda_1 = .001$
FFNN	*1.2201*	*Structure = [15, 15, 15, 15]* [11]*; dropout = .1; epochs = 16; learning rate = .001;* *hidden activations* $= \max(0, x)$*; batch size = 16; batch normalization between layers*
CNN	*1.3084*	*Conv. structure*[12] *= [8, 8]; conv. dropout = .1; 2D max pool size = (2,2); kernel shape = (4,4)* *flattening layer neurons = 25; merged neurons*[13] *= [15, 15, 15, 12]; merged dropout = .25;* *output neurons*[14] *= [25]; output dropout = .1; batch size = 32;* *learning rate = .001; epochs = 84; batch normalization between conv. and merged layers.*

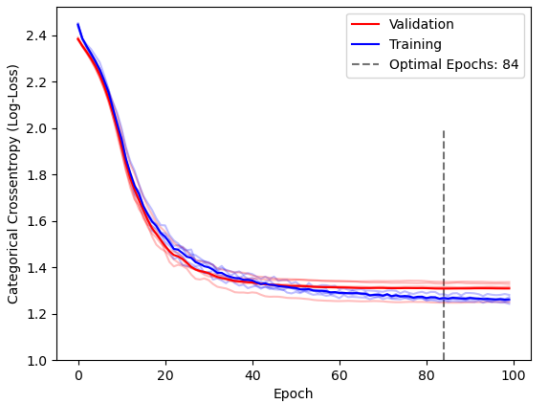
Importantly, for the neural networks, training was performed (within each fold) over an "excessive" number ($E_{max} = 100$) of epochs. However, validation/holdout loss was saved after each epoch, and once each of the $K = 5$ folds were fitted, validation/holdout loss was averaged across the fits at each $\ell = 1, 1, \ldots, N$. The optimal epoch number $\hat{E} \in \{1, \ldots E_{max}\}$ was then chosen to be that epoch which had the best average validation loss across the folds. In this way, the neural networks were able to accommodate the K-fold cross validation process, while also avoiding overfit from too many epochs.

Figure 3: Neural Network Training/Validation Losses

(a) FFNN Training/Validation Loss



(b) CNN Training/Validation Loss



5 Results and Discussion

Using the most performant tuning settings for each model, I then generated test set (i.e. races in 2019-20) predictions, iteratively retraining the models ahead of each race. Cross entropy loss was my principal performance measure, though for fun, I also calculated: a modified Spearman correlation ("semi-Spearman") between most probable finishing position and finishing position[15]; the eleven category[16] prediction accuracy ("accuracy") as classified by a driver's most probable finishing position; and the proportion of test races in which the driver with the highest predicted winning probability indeed won ("P1 Accuracy"). Test results for these models are shown in Table I.

As evidenced by the cross entropy losses and semi-Spearman correlations, the multinomial logistic and feed-forward neural network – the two "simpler" machine learning algorithms – performed best on the test set. I suspect this was for three reasons.

[9] See the 'tuning' folder for full grid specifications.

[10] Cross entropy is calculated via $-\sum_{i\in\{1,\ldots,n\}}\sum_{k\in K} p_k^{(i)} \log \hat{p}_k^{(i)}$, where $i = 1, \ldots, n$ are the data samples, $k = 1, \ldots, K$ are the predicted categories, $p^{(i)} = \langle p_1^{(i)}, \ldots, p_K^{(i)} \rangle$ are the observed (one-hot) categories, and $\hat{p}^{(i)} = \langle \hat{p}_1^{(i)}, \ldots, \hat{p}_K^{(i)} \rangle$ are the predicted class probabilities.

[15] Note that finishers in the 11^+ category were denoted as finishing 11^{th} for the purpose of this correlation. In this way, this metric was a modified-Spearman correlation.

[16] i.e. P1-P10, P11$^+$.

Table 2: Test Set Results (2019-2020 Races)

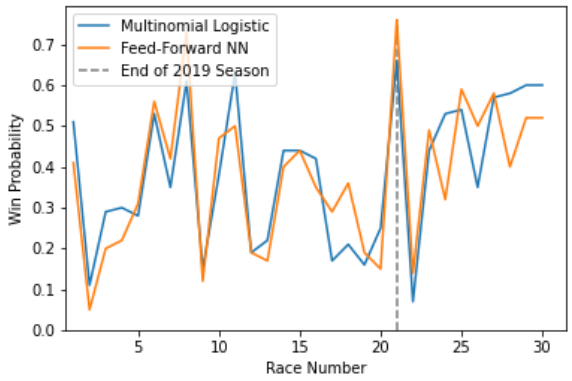
Model	Cross Entropy	Semi-Spearman	Acc.	P1 Acc.
Naive	1.2182	0.6000	0.5450	0.5000
MLR	1.1954	0.6178	0.5550	0.4667
FFNN	1.1909	0.6042	0.5417	0.4667
CNN	1.2439	0.5935	0.5433	0.5000

First, during the 2019-2020 testing period, the sport was overwhelmingly dominated by one team/driver (Mercedes/Lewis Hamilton), with the rest of the field following in a three-tiered, well-delineated hierarchy of decreasing performance.[17] Here, the "obvious" picks for these races – i.e. those that conformed to this clearly-defined hierarchy – were often the best ones, and as such, the simpler models were positioned to test well. Second, I suspect that the CNN's bifurcated structure precluded the convolutional layering from learning important interactions between the performance delta matrices and relevant external factors. Since these external factors (e.g. circuit features) were concatenated after CNN architecture had processed the performance deltas, the CNN filters may have been blind to critical features. Third, there were only 246 Grands Prix in the training dataset, leaving the CNN with few training instances. By contrast, the MLR and FFNN had $246 \cdot 24 = 5,904$ such instances (a training instance for each driver and race, instead of just a training instance for each race), which may also explain the CNN's inferior ability to generalize.

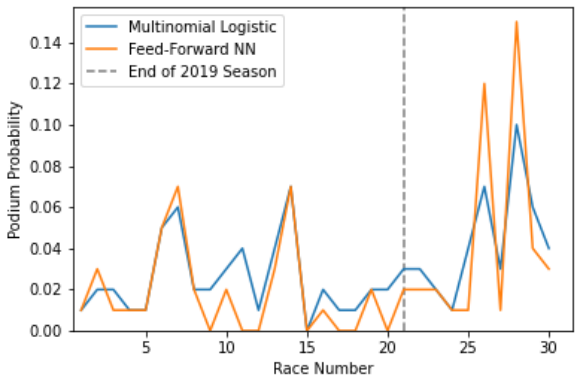
Nevertheless, it is worthwhile to inspect and compare model predictions from the test set. For example, consider the following probability predictions, as given by the two best-performing models, the MLR and the FFNN:

Figure 4: Assorted Driver/Race Predictions, 2019-2020

(a) Lewis Hamilton Win Probabilities, 2019-2020



(b) Daniel Ricciardo Podium Probabilities, 2019-2020



Here, we see that the two most performant models, the MLR and the FFNN, generally align in their predictions. However, they do have their disagreements – for instance, the FFNN is (occasionally) more optimistic in 2020 about Ricciardo's podium chances, while also more bearish about Hamilton's chances of winning early in the 2019 season.

6 Conclusions and Future Work

In this paper, I examined how three ubiquitous machine learning algorithms – to wit, multinomial logistic regression, feed-forward neural networks, and convolutional neural networks – can be leveraged to predict the outcomes of Formula One races. Under my data processing and feature engineering pipeline, I subsequently showed that the feed-forward neural network performed best, achieving a cross entropy of 1.909 on a test set of 2019 and 2020 Grands Prix.

However, this analysis is far from complete. With additional time and computing power, I would pursue a variety of leads in order to improve the model. Among other things, this would include: (1) examination of additional and/or more granular features; (2) more granular hyperparameter tuning for the aforementioned models; (3) consideration of other machine learning algorithms, including Long Short Term Memory Networks (which might better capture temporal dependencies within the data), tree-based classifiers, and gradient-boosting; and (4) potential ensembling of existing models. In the same way that the naive model (consisting of sample probabilities from previous Grands Prix) served as a baseline for the feed-forward neural network, the feed-forward neural network should serve as a baseline for future race prediction models.

[17] These less competitive seasons likely explain why the test cross entropy was less than the average validation cross entropy: because the test set's aforementioned hierarchy led to less competitive racing, Grand Prix results were less "scrambled," and test set included less inherent noise and was predicted more easily.

Works Cited

Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network." *2017 International Conference on Engineering and Technology (ICET)*. Antalya: IEEE, Aug. 2017. 1–6.

"CS231: Convolutional Neural Networks for Visual Recognition." *Stanford CS231*. publisher: Stanford University 16 Nov. 2020 <https://cs231n.github.io/convolutional-networks/#conv>.

"Ergast Developer API." *Ergast Developer API*. 16 Nov. 2020 <https://ergast.com/mrd/>.

"F1 predictor – Machine-learning based F1 race prediction engine." 16 Nov. 2020 <https://www.f1-predictor.com/>.

"F1Metrics." *F1 Metrics*. 16 Nov. 2020 <https://f1metrics.wordpress.com/>.

"Formula 1 Case Study." *Amazon Web Services, Inc.* 16 Nov. 2020 <https://aws.amazon.com/solutions/case-studies/formula-one/>.

Jing Yang and Guanci Yang. "Modified Convolutional Neural Network Based on Dropout and the Stochastic Gradient Descent Optimizer." *Algorithms* 11.3 (Mar. 2018): 28 16 Nov. 2020 <http://www.mdpi.com/1999-4893/11/3/28>.

Keras. "Keras documentation: Keras API reference." 17 Nov. 2020 <https://keras.io/api/>.

Lenail, Alex. "NN SVG." June 2020 17 Nov. 2020 <http://alexlenail.me/NN-SVG/index.html>.

Ma, Tengyu and Andrew Ng. "CS229 Lecture Notes: Supervised Learning." Sept. 2020. pp. 24-25 publisher: Stanford University. <http://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes1.pdf>.

Ma, Tengyu, et al. "CS229 Lecture Notes: Deep Learning." Oct. 2020. publisher: Stanford University. <http://cs229.stanford.edu/notes2020fall/notes2020fall/deep_learning_notes.pdf>.

Moore, Justin. "Who's The Best Formula One Driver Of All Time?" *FiveThirtyEight*. May 2018 16 Nov. 2020 <https://fivethirtyeight.com/features/formula-one-racing/>.

Mozer, Michael C., et al. *Proceedings of the 1993 Connectionist Models Summer School*. Google-Books-ID: 0SEBAwAAQBAJ pp. 335-336. Psychology Press, Mar. 2014.

"Supervised learning — scikit-learn 0.23.2 documentation." 17 Nov. 2020 <https://scikit-learn.org/stable/supervised_learning.html#supervised-learning>.

"The fastest driver in Formula 1." *Amazon Web Services*. Aug. 2020 16 Nov. 2020 <https://aws.amazon.com/blogs/machine-learning/the-fastest-driver-in-formula-1/>.

"Unsupervised Feature Learning and Deep Learning Tutorial." publisher: Stanford University 16 Nov. 2020 <http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/>.

Xiong, Jian, Kai Zhang, and Hao Zhang. "A Vibrating Mechanism to Prevent Neural Networks from Overfitting." *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. Tangier, Morocco: IEEE, June 2019. 1737–1742.