FINDING OPTIMAL RACING LINE COMPONENTS WITH MACHINE LEARNING

by
Joshua David Christensen

A Senior Honors Thesis Submitted to the Faculty of
The University of Utah
In Partial Fulfillment of the Requirements for the

Honors Degree in Bachelor of Science

In

The School of Computing

Approved:

H. James de St. Germain Thesis Faculty Advisor	Mike Kirby Assoc. Director, School of Computing
Erin Parker Honors Faculty Advisor	Sylvia D. Torti, PhD Dean, Honors College

April, 2019

ABSTRACT

In Motorsports, there exists a concept known as the optimal racing line. The optimal racing line is the racing line that, if followed, should allow for the minimum possible lap times. A substantial amount of research has surrounded finding this optimal line as a combination of two constructs: the Minimum Curvature Path (MCP), and the Shortest Path (SP). Both of these component paths can be found through traditional optimization methods on closed loop courses. We aim to train a machine learning model to predict the shortest path on short sections of road. By generating these paths for a set of 95 real-world racetracks, and using the results as ground truth to train machine learning models, this paper trains and evaluates a number of different neural networks to predict the shortest path. While the resulting networks were ultimately not accurate enough to be used in place of the traditional optimization methods, they may find use in establishing more narrow boundaries for the traditional optimization processes.

Contents

1 BACKGROUND

In Motorsports, there's a concept known as the 'optimal racing line'. Generally, the 'racing line' refers simply to the path the racecar takes around the track. The 'optimal' racing line is the racing line that, if followed, should impart the minimum possible lap times. Finding the optimal racing line is a difficult problem: the speed and path of the car must be perfectly balanced. A huge number of factors can impact the geometry of the optimal line: tire traction, available acceleration, vehicle dynamics, and track conditions can all impact the racing line[1]. A large amount of research has surrounded finding this 'optimal line'[1, 2, 3].

1.1 THE BASICS OF THE OPTIMAL RACING LINE

Primarily, the optimal racing line problem surrounds the question of choosing between the shortest path on the track, and the path throughout the track that provides the least amount of overall lateral acceleration if followed at a fixed speed. The ideal combination of the shortest path and this 'least curvy' path should provide the ideal balance of speed and path length that should allow a driver to minimize their lap time[3]. A method for finding both of these paths is described in Braghin, Cheli, Melzi, *et al.* [3], that will be paraphrased here. In Braghin, Cheli, Melzi, *et al.* [3], this path imparting the least amount of lateral acceleration is referred to as the 'lowest curvature trajectory'. This is because, when driving around a racecourse at a given speed, the lateral acceleration the vehicle experiences is defined by how tightly the vehicle turns. When traveling around a corner, this lateral acceleration can be viewed as the centripetal acceleration a_c experienced when traveling around an arc of

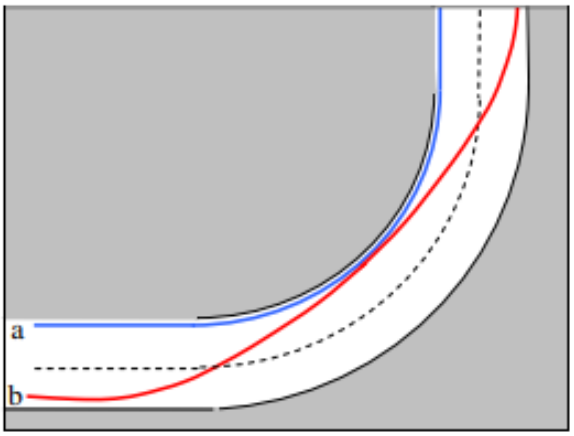


Figure 1.1: The shortest path around the course (a) and the 'least curvy' path around the course (b).[3]

radius r, at a velocity v. From basic physics, the equation relating these variables is as follows:

$$a_c = \frac{v^2}{r} \tag{1.1}$$

From this equation, if velocity v is held constant, the only way to minimize the lateral acceleration a_c is to maximize the radius r. both Braghin, Cheli, Melzi, *et al.* [3], as well as this paper, use the sum of the lateral acceleration throughout the course as a basis for finding this "least curvy" path. A diagram comparing the two paths, from Braghin, Cheli, Melzi, *et al.* [3], can be seen in Figure 1.1

1.2 DEFINING TRACK SEGMENTS

In order for either path to be created, first a framework to describe such paths must be established. The course must be split up into sections defined by lines perpendicular to the boundaries of the course. The points on this path can be defined as a distance (0,1) along the line. This framework is described in Figure 1.2. Line i can be defined by some function $f_i(\alpha)$ that maps $R \mapsto R_2$, where the function outputs are the point P_i's coordinates in global (X,Y) space. These functions take the form $f_i(\alpha) = \mathbf{r}_i\alpha + \mathbf{d}_i$, where r_i is an (X,Y) unit vector pointing in the direction of the path, from the inside of the track out, and p_i is

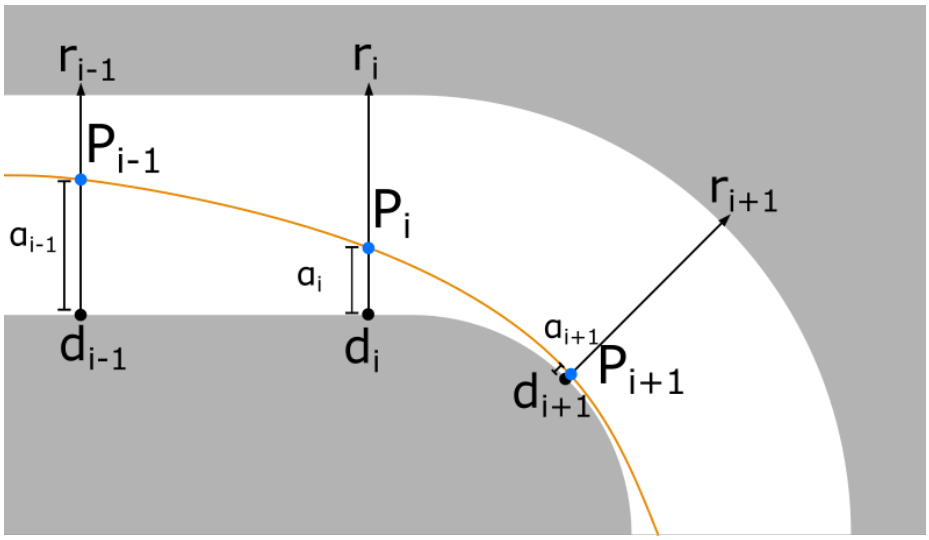


Figure 1.2: The racing line description framework used in Braghin, Cheli, Melzi, *et al.* [3]

an (X,Y) vector specifying the intersection between the inner curve of the track and line i. Therefore, each point P_i can be defined by:

$$\mathbf{P}_i = f_i(\alpha_i) = \mathbf{r}_i\alpha_i + \mathbf{d}_i$$

The racing line that we derive is a a cubic spline, interpolated through each point (X_i, Y_i). Cubic splines are used because they provide the minimum number of degrees of freedom required to specify the position of the car over time, as well as the ease with which they can be converted to vehicle steering inputs[4]. The splines are in the form:

$$\begin{cases} x_i(t) = a_{i,x} + b_{i,x}t + c_{i,x}t^2 + d_{i,x}t^3 \\ y_i(t) = a_{i,y} + b_{i,y}t + c_{i,y}t^2 + d_{i,y}t^3 \\ t(s) = \frac{s - s_{i0}}{ds_i} \end{cases}$$

(Equations pulled from [3])

1.3 FINDING THE SHORTEST PATH

The procedure for finding the shortest path, as described in Braghin, Cheli, Melzi, *et al.* [3], is fairly straightforward. We discard the cubic spline interpolation for computational reasons, rather opting to approximate the length of the path by finding the Cartesian distance between each point $\mathbf{P}_i = (X_i, Y_i)$. This can be thought of as drawing straight lines between each point P_i, and then finding the overall path length by finding the sum of the length of each line. Using our established variables α_i as independent variables (defined as $\bar{\mathbf{x}}$ in Braghin, Cheli, Melzi, *et al.* [3]), we simply minimize the following equation for path length L_p, where n is the number of points we have:

$$L_p = \sum_{i=1}^{n} \|\mathbf{P}_{i+1} - \mathbf{P}_i\| = \sum_{i=1}^{n} \|(\mathbf{r}_{i+1}\alpha_{i+1} + \mathbf{d}_{i+1}) - (\mathbf{r}_i\alpha_i + \mathbf{d}_i)\|$$

Braghin, Cheli, Melzi, *et al.* [3] goes in depth into the solution of this optimization problem. It is sufficient to say that from this, we can determine a set of points that, when interpolated on, give us the shortest possible path around the track.

1.4 FINDING THE 'LEAST CURVY' PATH

While Braghin, Cheli, Melzi, *et al.* [3] repeatedly use the term 'least curvy' path, it's more helpful to imagine this 'least curvy' path as the path that experiences the least amount of acceleration if followed with a fixed speed. If one examines the parameterization defined at the end of section 1.1, one can see that the x and y coordinates are indirectly parameterized to s, through an intermediate variable t. If we differentiate the x and y equations with respect to s, we can find some acceleration function $A(s)$ that returns an acceleration $\mathbf{a}_s$ for any given point s:

$$\mathbf{a}_s = A(s) = [\frac{d^2x(t(s))}{ds^2}, \frac{d^2y(t(s))}{ds^2}]$$

Since we use cubic spline interpolation, each spline from point P_i to point P_{i+1} is represented in the form of a polynomial with a degree of at max 3. The second derivative of a polynomial with a degree of at most 3 is a simple linear equation. If the equation for the original spline was:

$$x_i(t) = a_{i,x} + b_{i,x}t + c_{i,x}t^2 + d_{i,x}t^3$$

The equation for the second derivative of that spline would be:

$$x_i''(t) = 2c_{i,x} + 6d_{i,x}t$$

In order to minimize acceleration, one would only have to minimize $A(s)$, using the easily computed derivatives of our splines. Like the shortest path solution, this optimization problem is explored with more detail in Braghin, Cheli, Melzi, *et al.* [3].

1.5 SPECIFYING THE OPTIMAL COMBINATION OF PATHS

With both the shortest and 'least curvy' paths generated, the focus comes to finding the optimal combination between the two. We know that the fastest path is constrained by these two paths. No further speed increases can be attained by traveling outside the 'least curvy' path, as the radius of any given curve in the path has been maximized. No path length gains can be found, as our shortest path is precisely what the name implies. If the fastest path is constrained by these two paths, it follows that the fastest path's intersections with the track divisions described in section 1.1 must also be bound by the intersections of the shortest and least curvy paths. Taking advantage of that fact, we can quantify the proportion of each path we include in our final path using a simple coefficient ε.

As defined in the original Braghin, Cheli, Melzi, *et al.* [3], this was a simple global coefficient ε, such that, for each line i, the final α_{if} was equal to $\alpha_{if} = \varepsilon\alpha_{is} + (1-\varepsilon)\alpha_{ic}$. In this equation, α_{is} and α_{ic} were the alphas of the shortest, and least curvy paths, respectively. This successfully binds the new path in between our shortest and 'least curvy' paths. The

calculation of α_{if} is a complex problem in and of itself, and not within the scope of this paper.

1.6 THE LIMITATIONS OF PREDICTING SHORTEST PATH & MCP

As detailed above, Braghin, Cheli, Melzi, *et al.* [3] specifies techniques for finding the shortest paths and MCP for a set closed course. This proves extremely useful for simulation and analysis, but many real-world scenarios cannot assume such global knowledge. This presents a problem, as global knowledge is required to ensure our predictions for shortest path and MCP are accurate. Take the example where we have the next section of track, and we're trying to predict the shortest path. This is shown in Figure 1.3. In this case, the shortest path is represented in the form of a blue line, and the limits of our vision is represented as a red line.

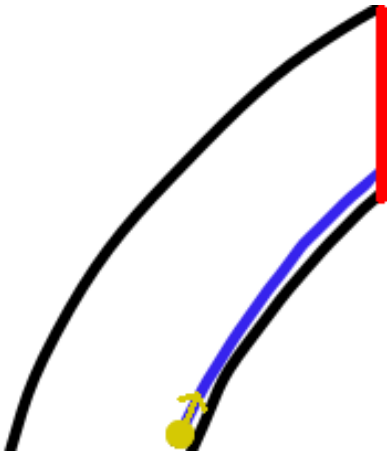
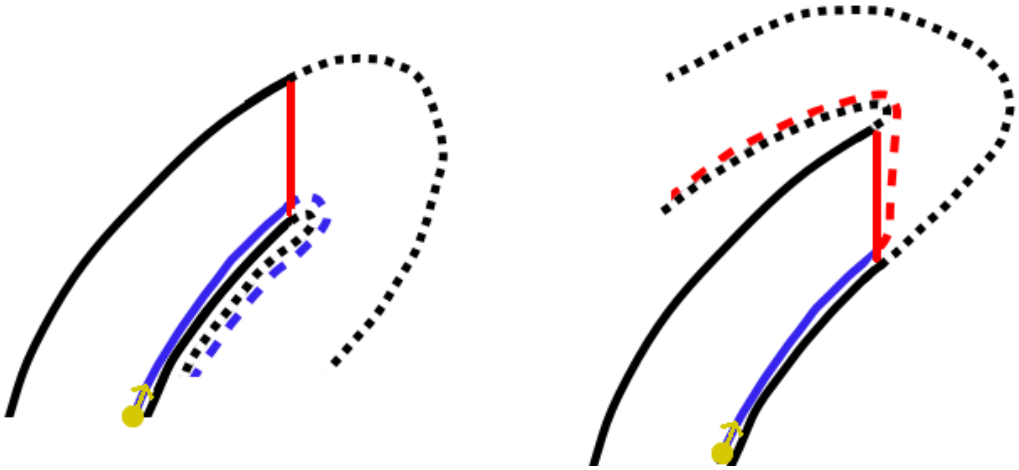


Figure 1.3: A short segment of track, and a reasonable prediction for the shortest path

This is a reasonable prediction for the shortest path. From a fundamental standpoint, a race course is a closed area bounded by an outer path and an inner path. For a track with a convex inner and outer path, the shortest path is equal to the inner path. It follows, then, that the inner path is a reasonable prediction for the shortest path for some unknown track segment. However, without global knowledge, it's impossible to know for sure. For

example, what follows in figures 1.4b and 1.4a are two examples for what could follow our short segment.



(a) Our prediction is valid (track is convex) (b) Our prediction is invalid (track is not convex)

Figure 1.4: Two potential track prediction outcomes

In figure 1.4a, our prediction appears to pan out. Our assumption about the convexity (or, at least, local convexity) of the course was correct. We were on the correct side of the turn, and were able to successfully hug the inner part of the course to find the shortest path. The flip side of this assumption can be viewed in figure 1.4b. In this example, our guess was completely wrong. We're forced to change our trajectory entirely, adding unnecessary distance to the path. This path would be completely useless for Braghin, Cheli, Melzi, *et al.* [3] style optimal pathfinding.

The dilemma presented here can be summarized as follows: We seem to need global track knowledge for MCP and SP estimations, but such information is not always available. However, despite the issues, a system that can make useful guesses without requiring the entire course to be mapped out could prove useful in practical applications. It's unclear how, or if, such a system can be built.

2 INTRODUCTION TO NEURAL NETWORKS AS A SP MODEL

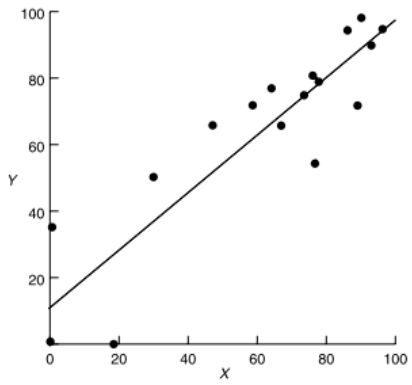
2.1 SP AND MCP AS A REGRESSION PROBLEM

In the pages that follow, neural networks will be evaluated as a potential solution to this prediction issue. Neural networks are uniquely poised to solve the MCP/SP prediction problem described earlier. Fundamentally, the MCP/SP issue is one of regression: we have a set of points in R^2 that define our binding paths, and we need to find another set of points in R^1 (the α values described in sections 1.3 and 1.4) that describe the MCP/SP. One can think of this as a more complex version of linear regression, where one is tasked with finding a function that best fits some set of points, of the form:

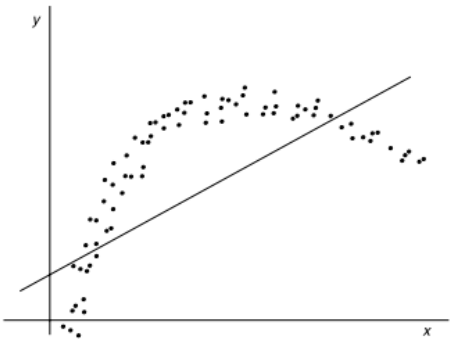
$$y = a_0 + a_1x_1 + a_2x_2^2 + a_3x_3^3 + ... \tag{2.1}$$

Where $X = \{x_1, x_2, x_3, ...\}$ are the inputs to the function, and $A = \{a_0, a_1, a_2, ...\}$ are the weights that one modifies in order to best fit the line. For an example, see Figure 2.1a, courtesy of Rencher and Schaalje [5, p. 130, Figure 6.1].

The regression problem at hand cannot use simple regression techniques. When using machine learning to mimic a function, including regression problems such as this, assumptions are inherently made about the space in which the function exists. Ideally, the function being targeted will exist inside the set of functions that the learning technique can generate (known as the 'hypothesis space'). For linear regression, the assumption is that the examples you're fitting were generated by a function that is linear, or close to linear. When this assumption is false, the target function is not within the linear hypothesis space, and as a



(a) An example of linear regression.



(b) An example of linear regression on non-linear examples.

Figure 2.1: Two examples from Rencher and Schaalje [5, p.130]

result, our line cannot effectively generate new examples (see Figure 2.1b).

2.2 ADVANTAGES AND STRUCTURE OF NEURAL NETWORKS

Neural networks are an advanced regression technique with an extremely large hypothesis space. Other regression techniques, such as linear or polynomial regression, restrict the hypothesis space to a narrow set of functions. Neural networks on the other hand, are comprised of tens, hundreds, or thousands upon thousands of linear functions, or 'neurons', sorted into layers. The input data serves as the inputs for the first layer. That is to say, the first layer of the neural network consists of some number of linear functions in the form of equation 2.1, with a term $a_i x_i$ for each value x_i in the input data. From here, layers are added, each comprising of some number of these 'neurons', with the output of the previous layer's equations set as their inputs. The final layer is in the same form as the previous layers, with a number of neurons equal to the number of values one wishes the neural network to return. For a graphical example, see Figure 2.2, courtesy of Hastie, Tibshirani, and Friedman [6, p.393]. In this example, the input data is the bottom blue $x_1 ... x_n$ layer, there's a single red layer of neurons $Z_1 ... Z_m$, and then the yellow output neurons $Y_1 ... Y_k$

When a neural network is trained on examples, the weights $A = \{a_0, a_1, a_2, ...\}$ for each neuron in the network are adjusted with the ultimate goal of turning the network into

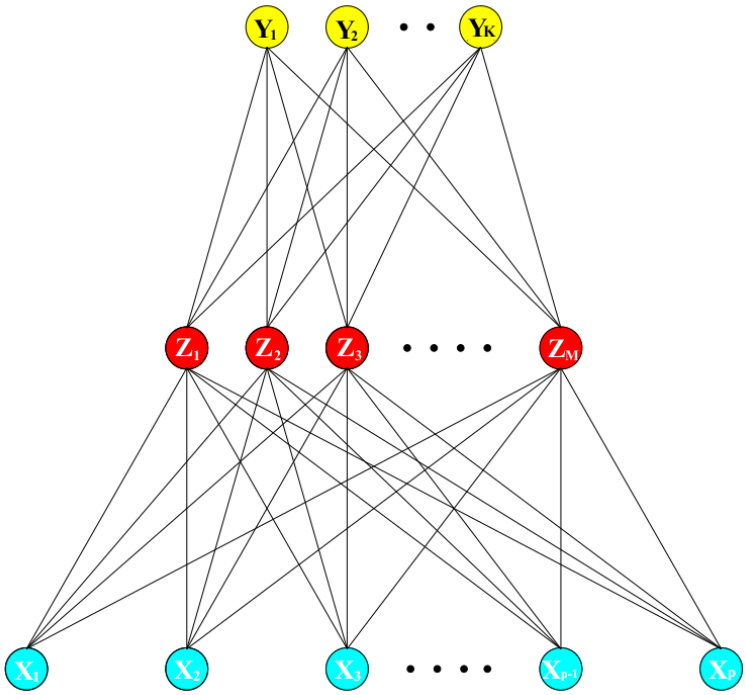


Figure 2.2: An representation of a neural network from Hastie, Tibshirani, and Friedman [6, p.393]

something the function that originally generated the examples. The fact that the network is comprised of several layers of linear functions allows it to mimic a wide variety of target functions, far beyond the limits of other kinds of regression[6, p.400]. In fact, part of the motivation behind the use of neural networks in certain applications is that they are able to find and use patterns not immediately apparent to humans. For all these reasons, neural networks are a prime choice in attempting to solve the MCP/SP regression problem.

3 METHODS

For this study, neural networks with fully connected (dense) layers were used. These are neural nets as described in section 2.2, where every neuron on a layer has as inputs every output on the previous layer. Neural networks can be extremely powerful, but their performance is highly dependent on the number of neurons per layer, and the number of layers used. Good choices for such parameters are difficult to find, as they are often problem specific. The work was done varying the number of neurons in order to attempt to find a viable neural network structure in order to learn MCP/SP patterns.

3.1 DATASET

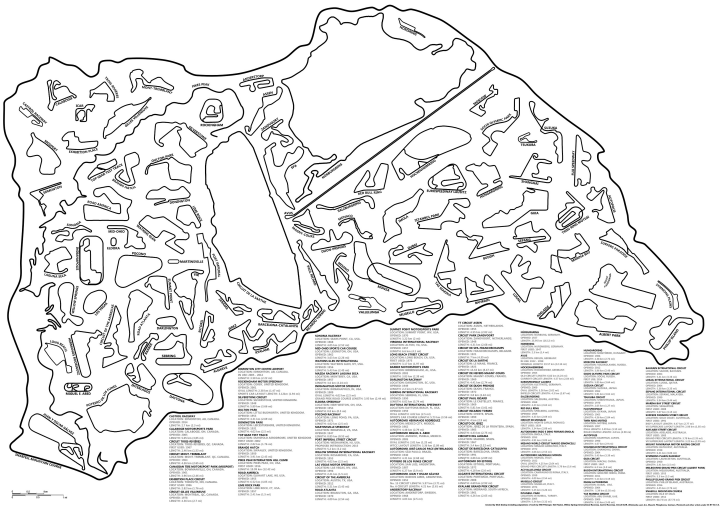


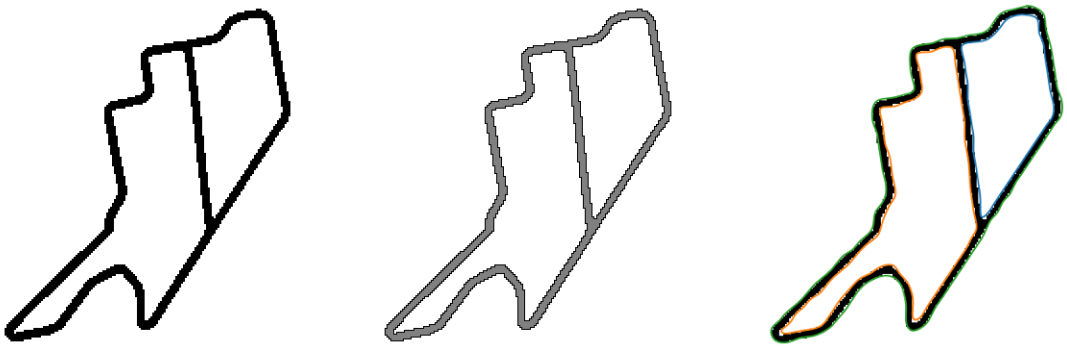
Figure 3.1: A poster, containing 95 world famous racing tracks.

In machine learning, a major issue is the availability of viable test data. In fact, the

vast majority of work performed for this project simply surrounded extracting and sanitizing test data for use in training. Because some of the advantages imparted by neural networks are related to patterns not obvious to humans, using data from arbitrary randomly generated tracks wasn't an optimal solution. Instead, data was extracted by applying image processing techniques to a scale representation of a number of famous race courses. The full representation can be seen in figure 3.1.

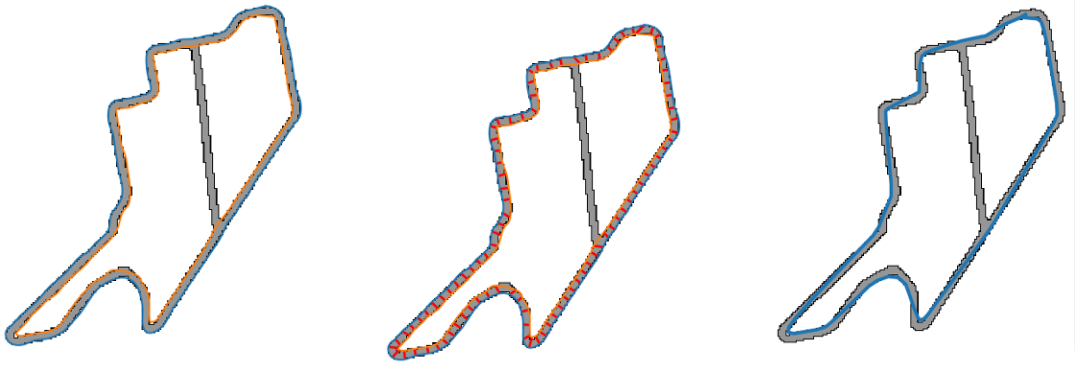
The tracks were separated out and then turned into vector representations appropriate for use with Braghin, Cheli, Melzi, *et al.* [3]-style path processing. The various steps of the image processing procedure are detailed in figures 3.2a through 3.3c. The general process for each track was as follows:

1. Isolate a single track from the image (Figure 3.2a).
2. Determine the boundaries of the track, in terms of pixels. (Figure 3.2b).
3. Determine the boundaries of the track in terms of cubic splines (Figure 3.2c).
4. Find a single canonical path through the course, in our case governed by the outer path. (Figure 3.3a).
5. Split the canonical path into sections, as per Braghin, Cheli, Melzi, *et al.* [3] (Figure 3.3b)
6. Using the sectioned track, run an adapted version of the optimization algorithm specified in Braghin, Cheli, Melzi, *et al.* [3] to find a shortest path (Figure 3.3c).



(a) The isolated track (b) Boundaries established (c) Multiple vector paths

Figure 3.2: Track processing steps, isolation through multi-path vectorization



(a) A canonical single path through the course (b) Broken into sections (c) The shortest path found for the track

Figure 3.3: Track processing steps, finding a canonical path through shortest path calculation

Full sized versions of these diagrams can be found in section 6. Due to the fact that the built tracks were comprised of an independent inner and outer boundary, as opposed to a path-defining centerline with equidistant inner and outer boundaries, [3] style shortest/MCP optimization techniques could not be used directly. The aforementioned techniques rely on the path being defined by a centerline, with inner and outer borders being equally spaced on either side of this defining line. This allowed the authors of [3] to parameterize their segments to a centerline, providing allowing them to simplify their loss functions to quadratic optimizations. As the segmentizing process developed for this project didn't impart such advantages, the loss functions used to generate our ground truth data were slightly different. The loss functions for SP and MCP are as follows, where alpha is the set of input scalars

as described in 1.1, and the notation $s_{ix}(\alpha_j)$ and $s_{iy}(\alpha_j)$ signifies the evaluation of the jth element of alpha on the ith segment of a given course:

$$sploss(\alpha) = \sum_{i=1}^{\|track\|} abs(s_{ix}(\alpha_{i-1}) - s_{(i-1)x}(\alpha_i)) + abs(s_{iy}(\alpha_i) - s_{(i-1)y}(\alpha_{i-1})) \tag{3.1}$$

Since the optimization of the MCP depends on accelerations, a naive velocity calculation was used:

$$v_x(s_a, \alpha_a, s_b, \alpha_b) = (s_{ax}(\alpha_a) - s_{bx}(\alpha_b)) \tag{3.2}$$

$$v_y(s_a, \alpha_a, s_b, \alpha_b) = (s_{ay}(\alpha_a) - s_{by}(\alpha_b)) \tag{3.3}$$

$$\begin{aligned} mcploss(\alpha) = \sum_{i=1}^{\|track\|} & abs(v_x(s_{i+1}, \alpha_{i+1}, s_i, \alpha_i) - v_x(s_i, \alpha_i, s_{i-1}, \alpha i - 1)) \\ & + abs(v_y(s_{i+1}, \alpha_{i+1}, s_i, \alpha_i) - v_y(s_i, \alpha_i, s_{i-1}, \alpha i - 1)) \end{aligned} \tag{3.4}$$

Both of these loss functions were found experimentally, evaluating a variety of different potential loss functions over the test-tracks and observing which resulted in the most correct data. Quadratic, exponential, and absolute-value based loss functions were evaluated. Optimizations were performed with the scipy basinhopping method, using a Limited Memory BFGS Optimizer. The initial guess for the shortest path optimization was the inner path, which seemed to provide the best starting point for the optimization. The highest performing initial guess for the MCP optimization proved to be a triangle distribution with a mode of .8, starting the optimization close to the outer path, but not on the outer path, as this usually served as a local minima that inhibited the proper execution of the optimization methods.

3.2 FEATURES AND LABELS

With a viable set of tracks with accurate shortest path information, the next step is to determine the input features to be used in training the neural nets. All information for a neural net must be in the form of a vector of real numbers. Our track representation made this fairly straight forward. Each segment of the track, as seen in Figure 3.3b, is defined by a line segment that spans the inner path and outer path. This segment was stored in terms of the Cartesian coordinates of the two points that make up the line segment. Based upon our representation, the most convenient way to turn a track into features for a neural net was to take the binding coordinates for some set of segments, and store them into a vector. For example, a single section division (the red lines in Figure 3.3b) defined by the points $P_1 = (1,2), P_2 = (3,4)$ would be stored as the vector $v = [1,2,3,4]$. Since sequences of track segments were required to predict the shortest path, the vector representations for each segment were appended end to end to form our final feature array. For example, segment vectors $v_1 = [1,2,3,4]$ and $v_2 = [5,6,7,8]$ would be combined to form final vector $v_f = [1,2,3,4,5,6,7,8]$, which would then be used as input data for the neural net. Training the neural net requires labeled data, so for every input array v_f built from a set of segments S, there was also built a label array l_f that corresponded to the correct α_{sp} values for every segment in S. All 95 tracks were converted to this format, taking sets of 20 sequential track segments per data point, providing over twenty-seven thousand labeled data points in order to train the neural nets on.

3.3 SEARCH SPACE

As mentioned at the beginning of this section, the difficulty in training dense neural networks is primarily centered on finding good values for the number of layers, and the number of neurons per layer. Improper values can result in a network that completely fails to learn the target function, and provides guesses that are little better than random chance. Fur-

thermore, the problem-specific nature of these parameters means that experimentation is required in order to find good values. Such experimentation was performed on this project, examining a large number of different potential network structures.

Throughout this document, the layer sizes will be described in terms of their magnitude relative to the input data. This is referred to as a 'layer factor'. The length of our input data, as described in section 3.2, is 80: there are four data points per segment, and 20 segments per test example. Therefore, a layer factor of [.25] is $\frac{1}{4}$ of 80, or 20 neurons. Layer sizes are described in this manner in order to more easily scale the structure of the neural network if the number of segments per example change. For example, if 30 segments per test example is decided to be the best choice for our features in the future, the hope is that the structure of the neural networks can remain essentially unchanged, with the layer sizes all scaling equally according to the increasing segments per example.

That being said, the investigation started off by examining the permutations of a number of layer scales, with network depth up to 4 layers. That is to say, we investigated neural networks with depth ranging from 1 layer to 4 layers, with layer factors chosen from the following list:

$$layer\_factors = \{0.25, 0.5, 2, 4, 6, 8\} \tag{3.5}$$

When a set of effective layer factors was found, further investigation was done into more networks of similar format. Accuracy was determined by the Tensorflow implementation. The Keras model was configured to set aside 20% of the input data for a validation set. That 20% was not used as training data, but was rather used to determine the performance of the network. All error rates included here are the validation error rates.

An aside, on notation: layer factors are read left to right, from input to output. A set of layer factors [2,1,3] would have the first layer have a layer factor of 2, the second a factor of 1, and the third a factor of 3.

4 RESULTS

4.1 DATA

Due to the intensive nature of the calculations required, this paper examines only the permutations of the learning factors, rather than dealing with any sort of replacement. Approximately 300 different runs were executed in the naive stage of the investigation. Layer sizes from 1 to 4, with a large variety of permutations run. As seen in Figure 4.1, the success of the different networks varied wildly.

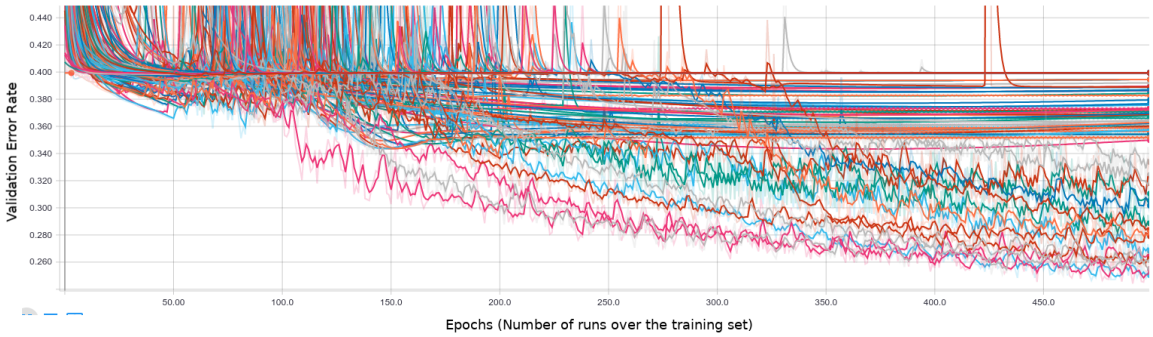


Figure 4.1: All of the error rates for our naive investigation

The vast majority of the networks tended to maintain error rates of approximately 0.4, which is simply unacceptable for a value that ranges between 0 and 1, and barely better than random chance. Analyzing this data in terms of final validation error, a few front runners emerged:

Layer Factors	Final Validation Accuracy
$[0.25, 4, 2, 0.5]$	0.247
$[0.25, 8, 6, 0.5]$	0.248
$[0.25, 4, 0.5, 6]$	0.254
$[0.25, 8, 2, 0.5]$	0.260
...	...

The results followed in this pattern, where networks with more layers and tighter starting layers prevailed. The top 8 results were all 4 layer networks, with initial layers of .25. From this, it was decided to further investigate networks of this format. Manually building and running networks with tight starting and ending layers, we were able to achieve error rates far better than the 0.4 standard, as seen in figure 4.2:

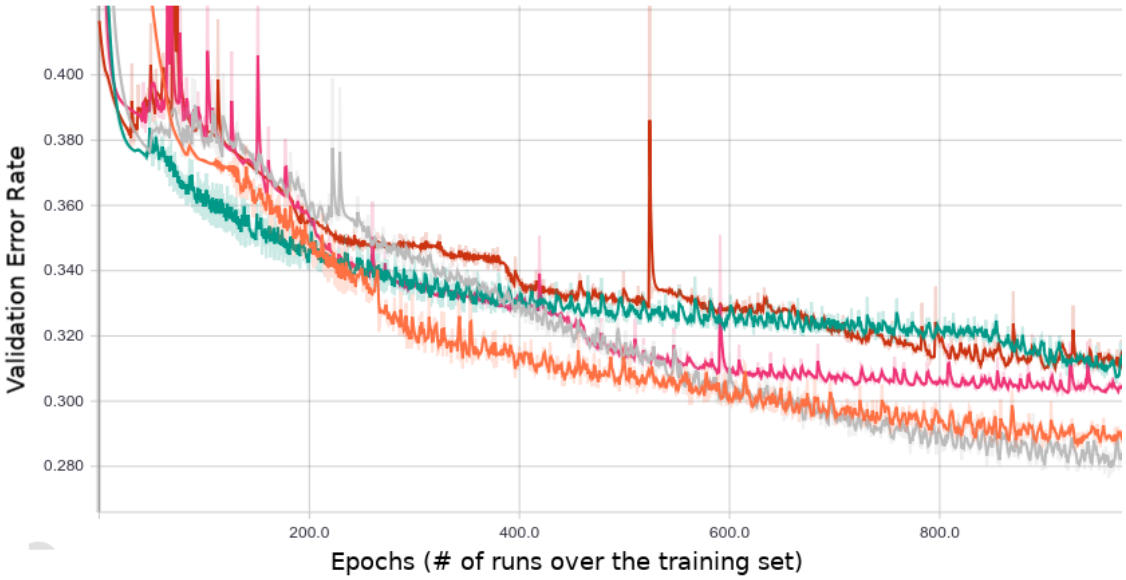


Figure 4.2: Various runs with bottlenecks at the beginning and end of the networks, as per 4.1

The brown, pink, orange and grey runs were all variants of networks that placed bottlenecks at the beginning and end of the net, and placed more neurons in the layers in the center. The teal was a slight variant on this format, where an additional bottleneck was placed in the center of the network. This did not ultimately prove a productive area of

Chart Color	Layer Factors
Brown	$[0.1, 4, 4, 0.1]$
Teal	$[0.1, 2, 0.1, 2, 0.1]$
Pink	$[0.1, 2, 2, 0.1]$
Orange	$[0.25, 4, 4, 0.25]$
Grey	$[0.1, 1, 1, 0.1]$

Table 4.1: Key for figure 4.2

investigation. A key for the graph can be found in Table 4.1.

The concept of bottlenecks improving the performance of the network prompted an exploration into a funnel-based network structure that has gradually decreasing layer sizes, from a layer scale of 1 to a final layer scale of .1. Such a network was investigated in figure 4.3, with a layer factor structure $[1, 0.8, 0.6, 0.4, 0.2, 0.1]$.

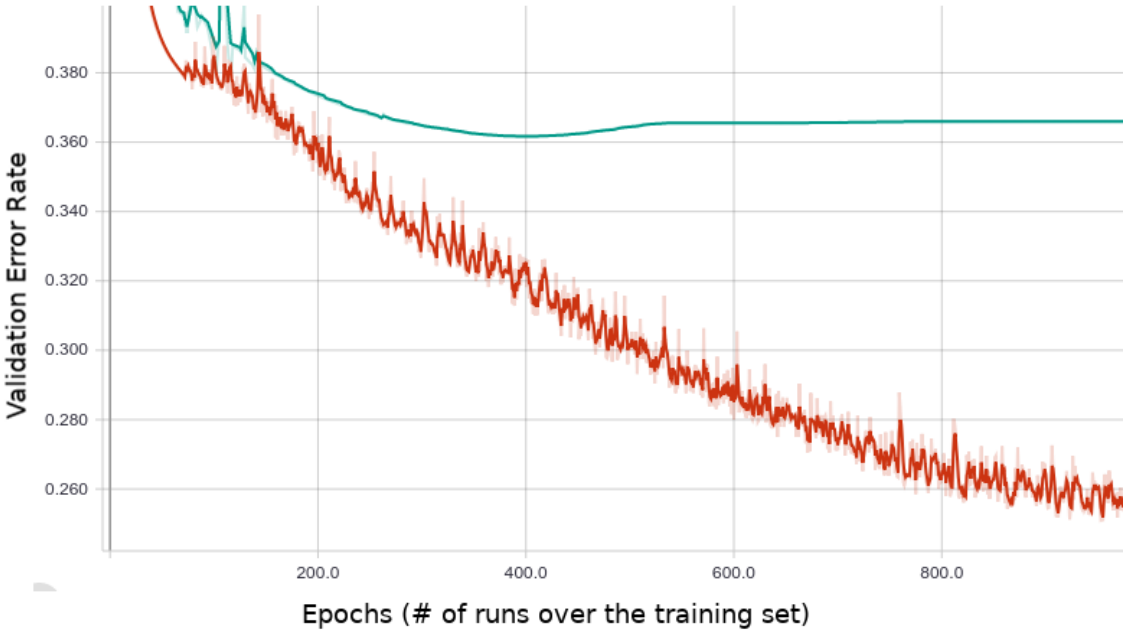


Figure 4.3: Two different runs for a layer setup of $[1, 0.8, 0.6, 0.4, 0.2, 0.1]$

Both runs in figure 4.3 are the same $[1, 0.8, 0.6, 0.4, 0.2, 0.1]$ layer factor structure, but the second run (teal) settles at a significantly higher error rate than the original run (brown). In order to attempt to get an idea of how repeatable these results are, ten iterations of the funnel network were run. For results, see figure 4.4.

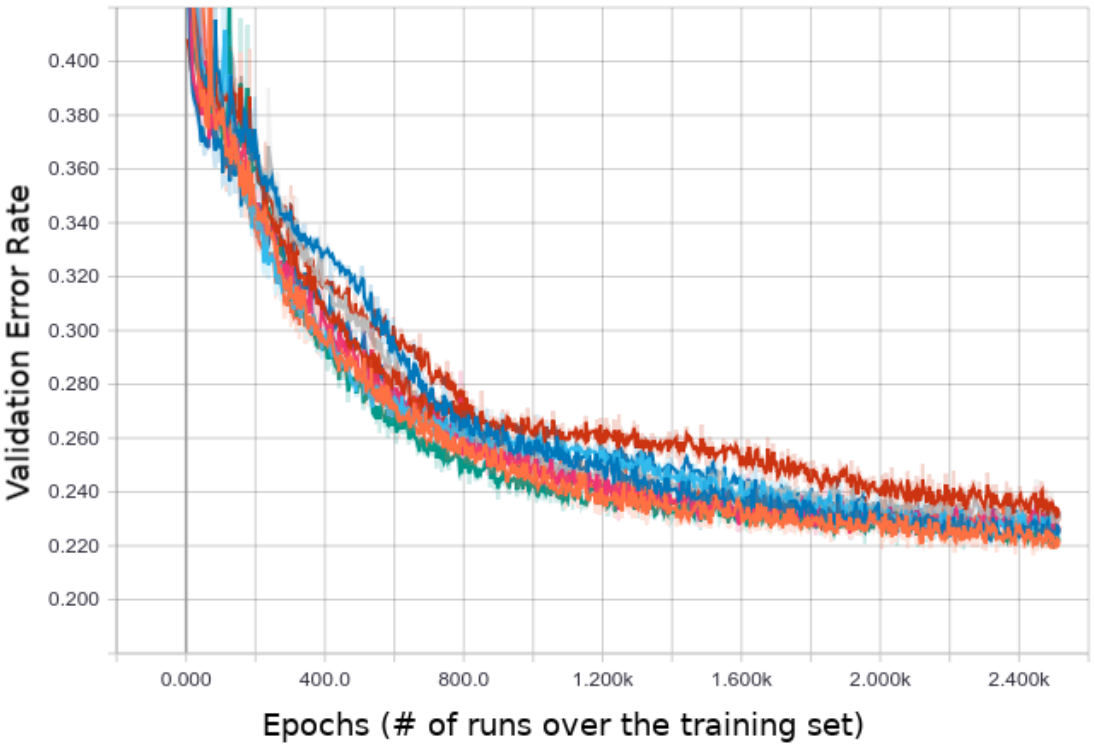


Figure 4.4: Ten different runs for a layer setup of $[1, .8, .6, .4, .2, .1]$

These results seem to imply that the failure experienced in figure 4.3 was an aberration.

5 CONCLUSION

5.1 ANALYSIS

Based upon the data collected in section 4.1, it seems safe to conclude that significant gains are achieved when predicting shortest path using neural networks that have bottlenecks that force layers to 'summarize' previous layers. From a conceptual standpoint, this makes sense. The structure of our features, described in section 3.2, generates four data points for every one data point in our result. Furthermore, it stands to reason that some of those points could very well include redundant information: horizontal paths will have nearly identical y coordinates for points on the inner and outer paths, for example. Forcing the network to only care about specific features of the input seems to increase the probability of successfully decreasing error rate.

5.2 EFFECTIVENESS OF NEURAL NETWORK MODELS

I'm not confident that this will prove to be a useful method in estimating MCP/SP. Even with the best techniques found, mean evaluation error rates still hovered around $\approx .25$, which is fairly severe for a number ranging $[0, 1]$. That shouldn't discount the potential usefulness of these networks. If further statistical analysis was performed on training data, it could be feasible to use the networks as rough outer bounds in a Braghin, Cheli, Melzi, *et al.* [3] style optimization problem. Since the SP and MCP simply serve to reduce the search space in Braghin, Cheli, Melzi, *et al.* [3], having slightly wider bounds for finding the optimal ε mentioned in section 1.5 should not prove fatal to the process.

5.3 OTHER USES OF SOFTWARE DEVELOPED

It is worth noting that the techniques developed in the execution of this document have been utilized in the DRAGLINE software system developed at the University of Utah. If the SP prediction models had performed better, they would have found application immediately in a software system designed to generate suggested racing lines and deliver them to drivers in training through an Augmented Reality interface. Due to the poor performance of the neural nets, this DRAGLINE software system instead leverages the optimization routines developed in service of this document. Additionally, this software system has the capability to record tracks in the sectionized format specified in this document, paving the way for future data-collection and subsequent development surrounding shortest and minimum curvature path prediction, as well as optimal pathfinding. The DRAGLINE software system also provides the unique opportunity to validate these paths in a real-world racing environment.

5.4 POTENTIAL FUTURE WORK

A major regret surrounding this investigation was that, due to the large amount of development overhead that surrounded even building the ground truth dataset (parsing, sectionizing and finding SP/MCP), the amount of time expended towards the actual goal of this paper was cut short. If future work were to be performed, a much wider variety of Neural Net architectures should be investigated. Suggested by Vivek Srikumar of the University of Utah School of Computing, LSTM Autoencoders may be a productive area of investigation, as the sequence-based nature of the segment data naturally lends itself to a sequence-based neural network architecture. This is supported by the observations in 5.1, that seemed to indicate that naive feedforward neural networks that had ”bottlenecks” to summarize previous layers performed best. As autoencoders are designed to generate condensed forms of their input data, this seems like a promising area of investigation.

6 FULL SIZE FIGURES

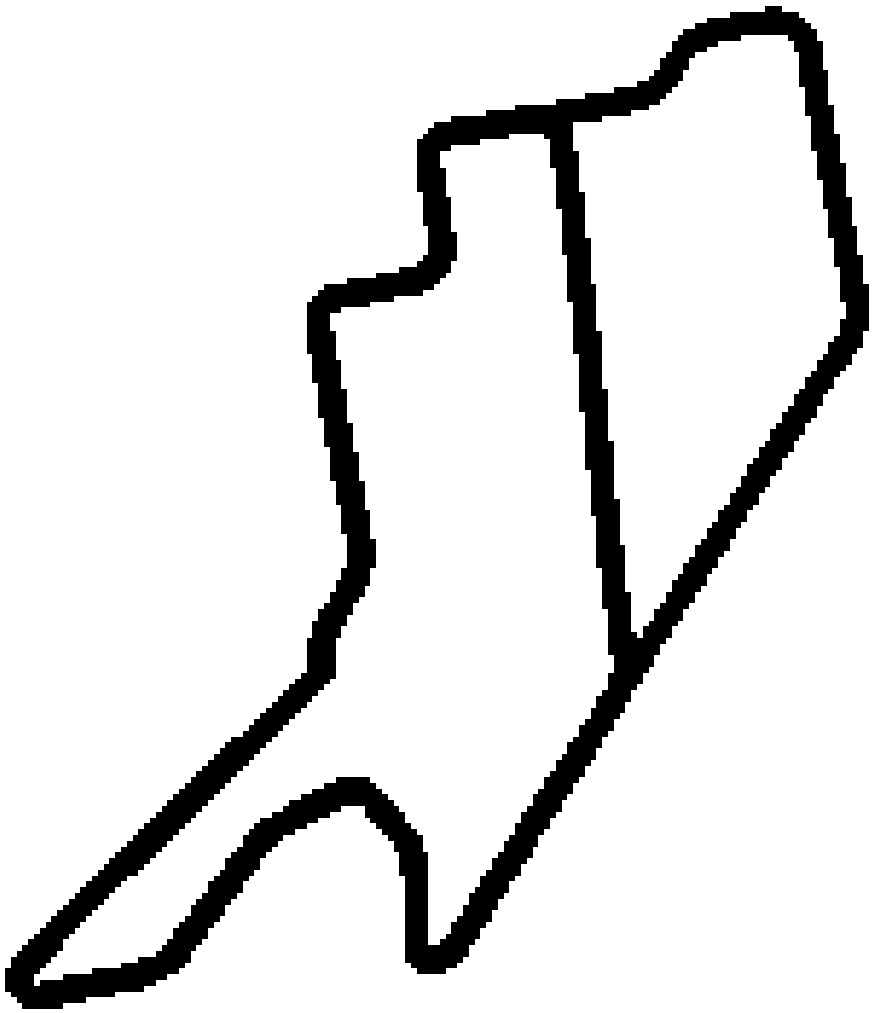


Figure 6.1: The isolated track

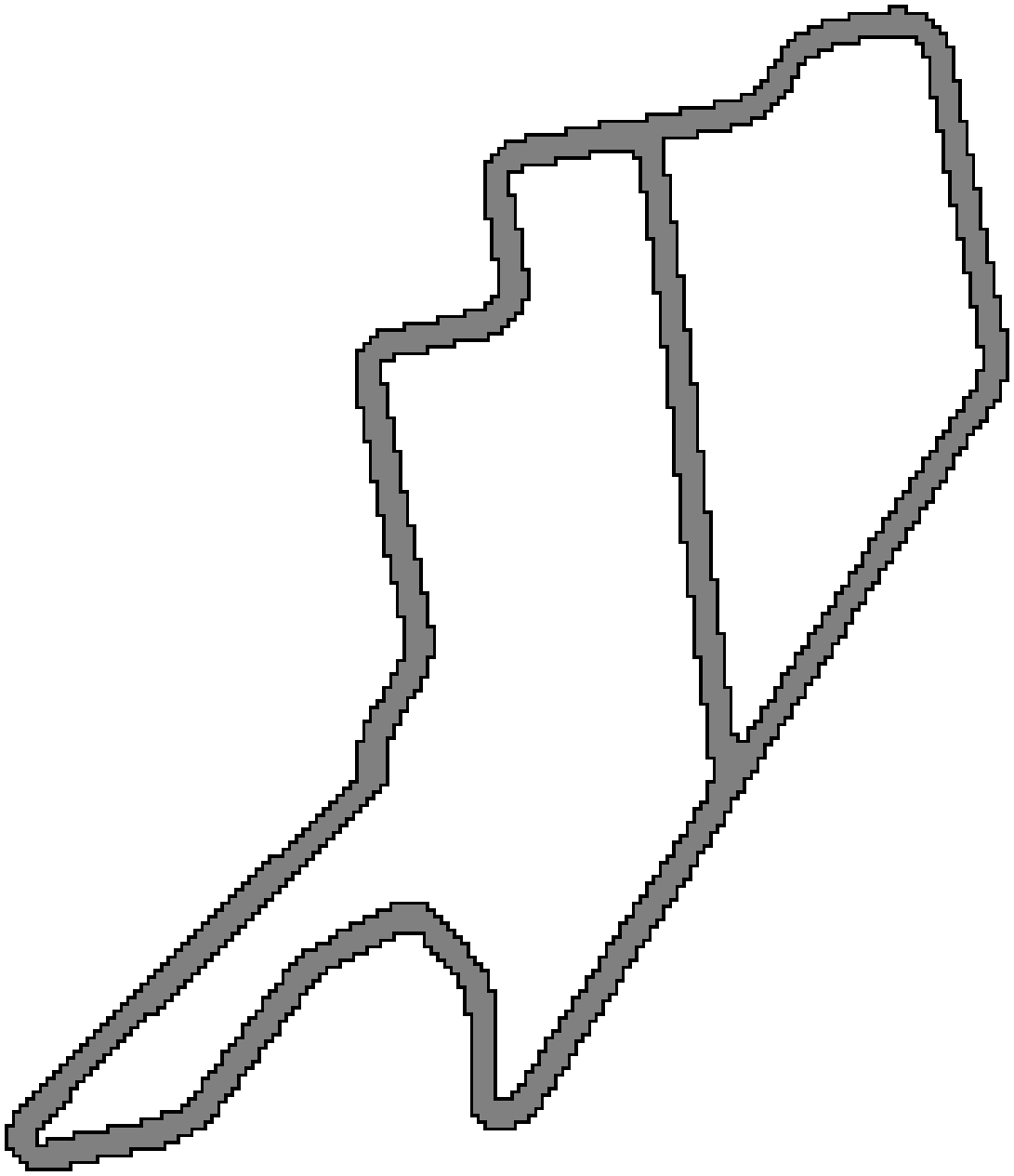


Figure 6.2: The track, with boundaries established

Figure 6.3: The vectorized track, with multiple paths

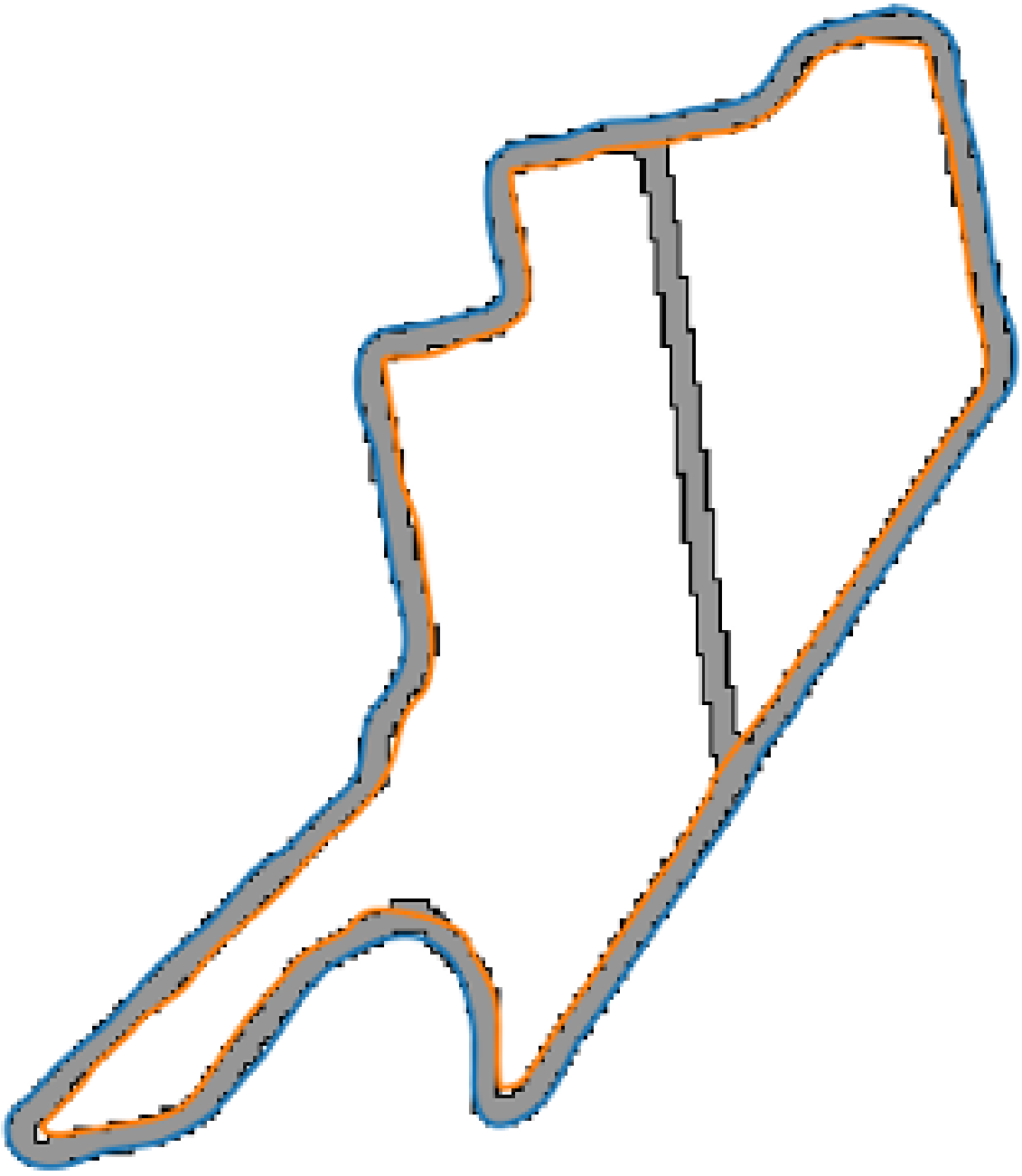


Figure 6.4: A canonical single path through the course

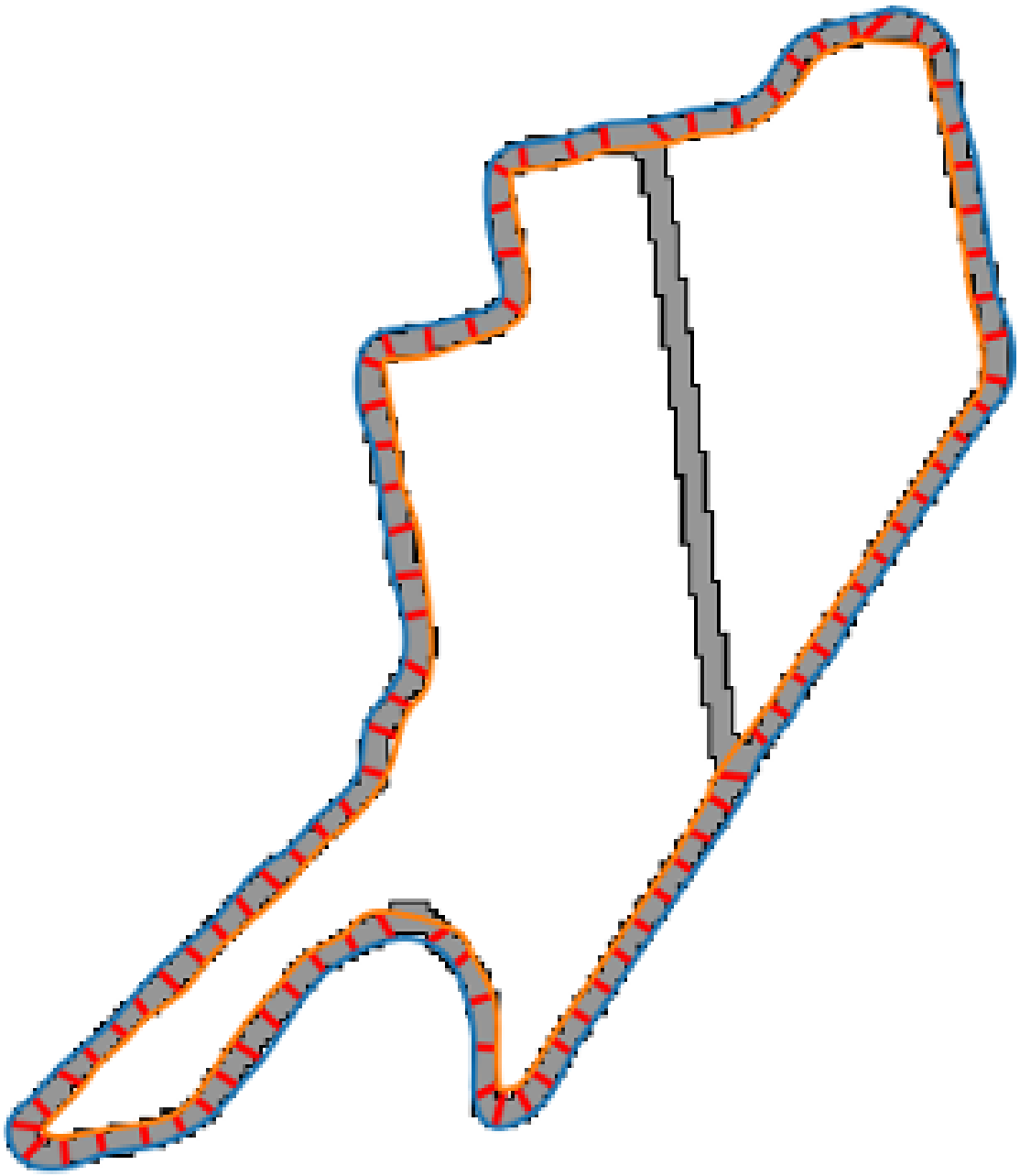


Figure 6.5: Broken into sections.

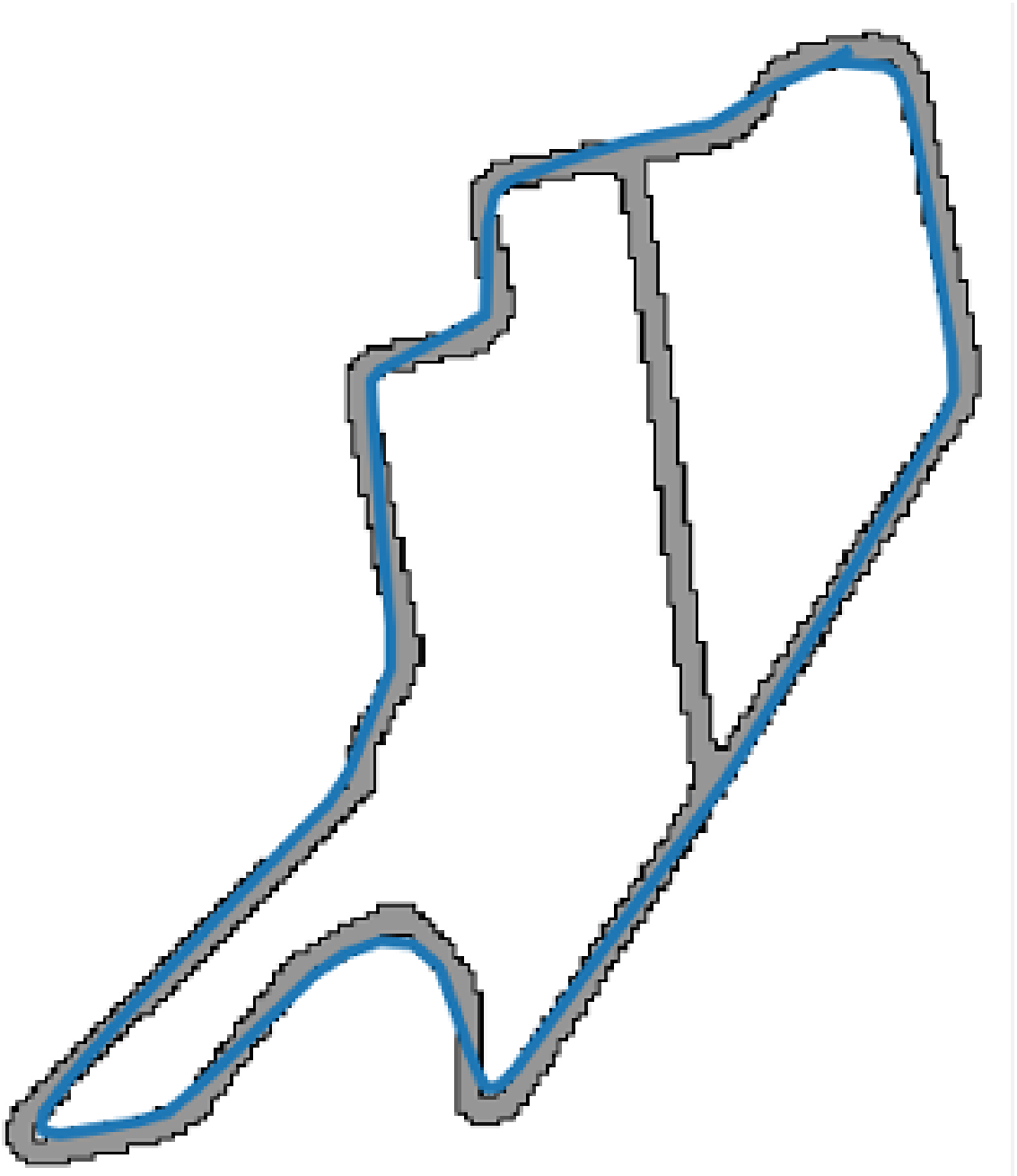


Figure 6.6: The shortest path found for the track

Figure 6.7: A poster, containing 95 world famous racing tracks.

Bibliography

[1] D. Casanova, "On minimum time vehicle manoevring: The theoretical optimum lap," PhD thesis, Cranfield University: College Rd, Wharley End, Bedford MK43 0AL, UK, 2001.

[2] L. Cardamone, D. Loiacono, P. L. Lanzi, and A. P. Bardelli, "Searching for the optimal racing line using genetic algorithms," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, Aug. 2010, pp. 388–394. DOI: 10.1109/ITW.2010.5593330.

[3] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni, "Race driver model," *Computers & Structures*, vol. 86, no. 13, pp. 1503–1516, 2008, Structural Optimization, ISSN: 0045-7949. DOI: https://doi.org/10.1016/j.compstruc.2007.04.028. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045794908000163.

[4] B. Nagy and A. Kelly, "Trajectory generation for car-like robots using cubic curvature polynomials," 1998.

[5] A. C. Rencher and G. B. Schaalje, *Linear models in statistics*. Wiley-Interscience, 2008.

[6] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning, Second Edition*. Springer, 2009.