AN ABSTRACT OF THE THESIS OF

Jaime L. Junell for the degree of Master of Science in Mechanical Engineering presented on July 9, 2009.

Title: Adaptive Methods for Robust Commercial Vehicle Control

Abstract approved: ______________________________

Kagan Tumer

This thesis explores the implementation of learning based control with predictive cruise control and the potential this technology has for increasing fuel efficiency while keeping on a well maintained schedule for commercial trucks. Traditional cruise control is wasteful when maintaining a constant velocity over rolling hills. Predictive cruise control is able to look ahead at future road conditions and solve for a cost effective course of action. Model based controllers have been implemented in this field but cannot accommodate all the complexities of a dynamic environment such as a stretch of highway in variable conditions. In this work, we focus on incorporating a learner into an already successful model based predictive cruise controller in order to improve its performance. We explore back propagating neural networks using several different input representations of varying complexity in order to predict future errors then take actions to prevent said errors from occurring. The results show that we are able to improve upon the model based predictive cruise controller by up to 60% average across the data. To obtain the best improvement it was found that highly descriptive inputs must be used in conjunction with training data that is very representative of the testing data. To obtain a more robust controller that can perform well on all terrain, use inputs that present less information and more generalizations to the neural network.

Adaptive Methods for Robust Commercial Vehicle Control

by

Jaime L. Junell

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented July 9, 2009
Commencement June 2010

ACKNOWLEDGEMENTS

My deepest thanks go out to Dr. Kagan Tumer for all his patience, guidance, and happy hours. Being part of the Adaptive Agents and Distributed Intelligence group has been a great experience for me all around. It is a privilege to be part of such an intelligent and fun group of people. I'd also like to acknowledge my fantastic committee members who have always been available to pass their wisdom onto me. Thanks so much to my sponsor, Daimler Trucks North America, for giving me this awesome opportunity to work with something interesting and applicable. Lastly, I'd like to thank my family who have always encouraged me in my endeavors.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

Chapter 1 – Introduction

With ever increasing oil prices and diminishing oil reserves, it has become both economical and environmentally necessary to preserve as much fuel as possible. One way to improve fuel efficiency while driving is to use the most effective driving techniques. For example, driving at lower speeds on the highway, braking smoothly and accelerating slowly are all ways to travel further on a gallon of gas. However, it is impractical for a human to compute the optimal position of their accelerator pedal to both save on fuel and still keep to the schedule they are on to get to their destination. One solution is to partially automate the vehicle's speed to optimize efficiency.

Cruise control, now standard or optional in most cars, was the first attempt to assist the driver. Conventional cruise control will help fuel efficiency on long flat stretches of highway by decreasing gas guzzling accelerations and decelerations. However, when the vehicle approaches an incline, the amount of fuel needed to maintain a constant speed is much greater. In this scenario, conventional cruise control can be very wasteful. One way to improve upon conventional cruise control is to use knowledge of the vehicles environment to adjust the vehicles speed for optimal fuel efficiency. If an incline is approaching then the vehicle can speed up before the hill so that it does not have to put so much gas into ascending that incline. This kind of technology is called Predictive Cruise Control(PCC) [23].

Predictive cruise control utilizes Global Positioning Systems (GPS) to obtain the changes in road elevation and curvature along the intended path of travel. Using this knowledge, it is possible to calculate the best velocity curve to optimize factors such as fuel economy and cost of time.

One of the largest industries to feel the fuel price increase is the trucking industry. Semi-Trucks are also notorious gas guzzlers due to their large size and weight. Although fuel rates are a big factor to trucking companies and independent truckers alike, a driver still wants to get to his/her destination in a timely manner. One of the challenges in optimizing fuel efficiency is finding a good balance between fuel consumption, and time constraints. Given a good model of the vehicle, its environment, and an objective function, an optimal solution can be found [23]. Herein lies the challenge. With such a complex system as a truck in a dynamic environment such as the open road, it is nearly impossible to produce an accurate model.

Traditional control methods are not often robust to inaccuracies in the model or unaccounted for sources of noise [37]. In the semi-truck example, such inaccuracies could be a result of miscalculated weight of the truck, slightly flat tires, wet or icy road conditions, high winds, and many more uncontrollable or unaccounted for factors. Adaptive control methods are much more robust to unknown functions [30]. As long as these unknown factors produce a continuous function, then a learning algorithm can find this trend despite not knowing the origin of the error [2]. This quality makes learning controls ideal for a scenario in the variable highway environment.

Daimler Trucks North America is one company that has decided to participate in PCC research. With a created objective function and a complicated model of a moving semi-truck, they were able to calculate a desired velocity curve to optimize their objective function. However, it has been found through empirical testing and data collection that the vehicle is not able to accurately follow the desired velocity. This error is likely due to variables unaccounted for in the mathematical model. Because of this error it is necessary to recalculate the desired velocity curve using PCC after a certain distance has been traveled by the vehicle. These recalculations are expensive and it is desired to minimize the number of recalculations needed by predicting the offset between the desired velocity and the actual velocity that the vehicle obtains. If the offset can be accurately predicted, then we will be able to make corrections to increase required recalculation distance.

The contribution of this thesis is to improve upon a model based control system by integrating a learning control method into the system. The adaptive part of the control improves the performance of the controller by accounting for imperfections in the model that cause error. This improvement will ultimately reduce errors in PCC, resulting in:

1. An increase in the controllers ability to reduce fuel consumption
2. A potential increase in the recalculation distance, thus reducing the processing power required.

In this thesis, we explore the use of neural network learning as applied to predictive cruise control.

Chapter 2 provides background information on recent works related to PCC and optimal fuel control. This chapter also briefly describes the different variables that can affect a moving vehicle on the highway and the important roles they play in velocity adjustment for optimal fuel efficiency.

In chapter 3, the problem definition and approach to the problem will be explained. Since we didn't know exactly what information or how much information would be needed to represent the function we were trying to learn, we approached the problem simply at first and then added complexity as needed. A simple solution would be best since it would be the quickest and least expensive to implement. Several input mappings were explored. We used different configurations rooted from the desired velocity curve from the PCC, and the elevation curve from the GPS.

In chapter 4, the most simple problem approaches are presented. What we have labeled "single concept" approaches, only use information from either the desired velocity curve or elevation data. We will see that even a very easy and simple neural network can show improvement in the system.

On the other hand, "Multi concept" approaches involve neural network inputs from both the desired velocity curve as well as the elevation curve. In chapter 5, two multi-concept approaches will be presented and the results discussed. We will see that giving the neural network more information can be beneficial or may yield no improvements.

Lastly, in chapter 6, conclusions will be drawn from the information presented in the previous chapters. Contributions of this thesis will also be discussed and possible future work in the area presented.

Chapter 2 – Related Research

In this chapter we will discuss the general field of autonomously driving vehicles and the technologies that have been developed in recent years. Then we will go into further detail of the predictive cruise control idea and the basic mathematics behind the model.

2.1 Advancements toward Autonomously Driving Vehicles

There have been several advances toward autonomously driving vehicles over the past decade. Many of the technologies developed are to further the comfort of driving or energy efficiency of the vehicle while maintaining or enhancing safety features. One of the more significant inventions is Adaptive Cruise Control (ACC). Traditional cruise control is a very common feature in modern vehicles and gives the driver the ability to take his/her foot off the pedal and still move at a constant speed that is inputted by the driver. Adaptive Cruise Control takes this idea to the next level of convenience. Vehicles equipped with ACC will automatically slow down when approaching a car directly in front of it. The vehicle with ACC will then maintain a safe following distance from the preceding vehicle. This is made possible by placing an ultrasonic or laser sensor in front of the vehicle [22, 29, 42]. This sensor will measure the distance between the vehicles. Paired with the values from the vehicle speedometer, an appropriate deceleration rate and following distance can be determined. Adaptive Cruise Control started with just control over the accelerator, but more current models can also control the brakes, therefore being able to avoid rear end collisions without any input from the driver [22, 32].

There are several suggested benefits to using ACC such as: It provides luxury and convenience to the fatigued driver, it can be used as a safety device if the driver is not paying attention, it helps reduce traffic congestion, and it improves gas efficiency by forcing slow accelerations and decelerations as traffic moves [5, 6, 22, 42]. One study claims that in a single lane, high speed (30m/s) scenario, traffic jams can be avoided if at least 20% of the vehicles are equipped with ACC [5]. Multiple lane and on ramp scenarios are more complex and the simulation results vary [6, 7].

The next most noteworthy technology being researched is predictive cruise control(PCC). In many ways, PCC is the next addition to the cruise control family after ACC. Traditional cruise control is able to increase gas efficiency on long flat highways, while ACC saves fuel in higher traffic scenarios. Adding PCC to the control system of the vehicle will help increase fuel efficiency by saving gas while driving through variable road conditions. However, the complete cruise control system is not ready yet. Predictive cruise control is still in the early years of research. A patent for the basic idea behind PCC was filed in 2002 by DaimlerChrysler AG and issued in 2006 [23]. A study into "Look Ahead Control to minimize fuel consumption" was conducted in Sweden in 2007 and utilizes a complex vehicle and road model within a control system using several numerical methods. Results of the study found a successful decrease in fuel usage [14].

2.2 Predictive Cruise Control

Predictive Cruise Control(PCC), utilizes knowledge of road conditions further along the intended path so that the vehicle can take actions that will decrease the total fuel needed for not only the present time, but also for several moments in the future. In traditional cruise control, a hill can be approaching but the vehicle will continue to go the same speed. In PCC, the fuel optimal action to take may be to increase speed before the hill and decrease speed during the climb up knowing that once the peak of the incline is reached, gravity may help to alleviate the energy required from the engine. Figure 2.1, shows the basic concept of PCC. In the figure we can see lower and upper bounds for the velocity. Having these boundaries prevent the vehicle from driving too fast and getting a speeding ticket, and driving too slow to where it doesn't meet the time restraints.

Although the models used in the current PCC algorithms are extremely complex with several factors, the basic mathematics are simple to explain and understand. To give the reader a very brief introduction to why PCC works, we will discuss the main forces that act on a moving vehicle. In Figure 2.2 we can see a free body diagram of these base forces. The major forces acting on the vehicle are drag force, force associated with the incline of the road, the force applied by the

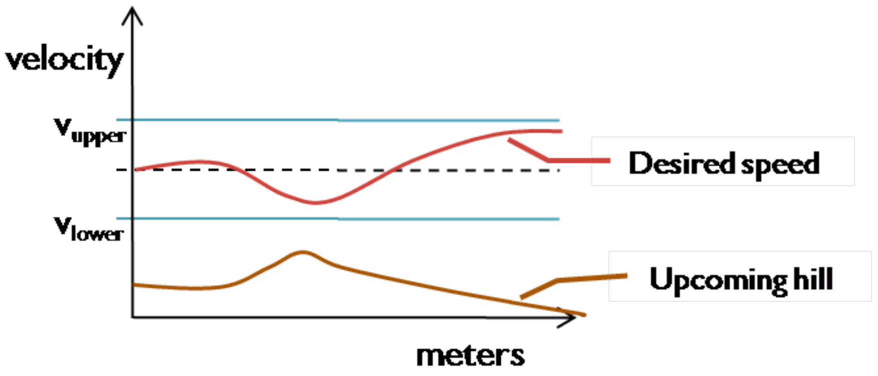

Figure 2.1: Predictive Cruise Control main concept: Given specified speed constraints, PCC system will solve for the optimal velocity profile for several meters ahead of the vehicle's current position.

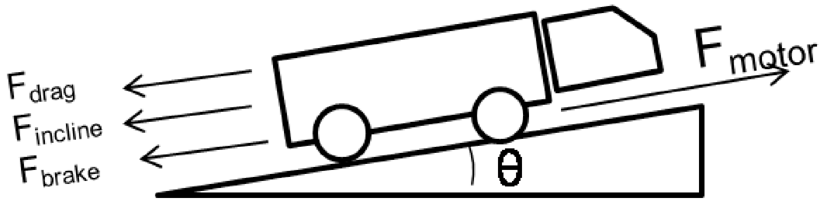

Figure 2.2: Forces on a vehicle: Several forces affect the vehicle as it moves

brake, and the force that results from the energy spent by the engine. Given these forces, the acceleration of the vehicle can be determined by Equation 2.1.

$$\sum F = m\frac{dv}{dt} = F_{motor} - F_{drag} - F_{incline} - F_{brake} \tag{2.1}$$

$$F_{motor} = \eta \frac{\tau}{r_{crankshaft}} \tag{2.2}$$

$$F_{drag} = (\frac{1}{2}c_d A \rho_{air})(v + wind)^2 \tag{2.3}$$

$$F_{incline} = mg\sin\theta \tag{2.4}$$

$$F_{brake} = 0 \tag{2.5}$$

Equations 2.2 - 2.5 give more detailed calculations of the forces on the vehicle. We can see that the force required by the motor increases as the drag force, incline, brake force and acceleration of the vehicle increase. As seen in Equation 2.5, we assume that the force of the brake is zero because similar to conventional cruise control, if the driver taps the service brakes, then cruise control will turn off. The incline affects $F_{incline}$ by changing the angle Θ and therefore, changing the affect that gravity has on the vehicle(Equation 2.4). Figure 2.3(Left) shows how the incline of the hill affects the associated force. The drag force is mostly effected by the changing velocity of the vehicle but can also be effected by wind. Equation 2.3 gives the equation for drag force, where $c_d A$ is a constant for the vehicle and ρ_{air} is the density of air at sea level and at 20°C. In Figure 2.3(Right), wind is neglected and the relationship between drag and velocity of the vehicle is shown [12].

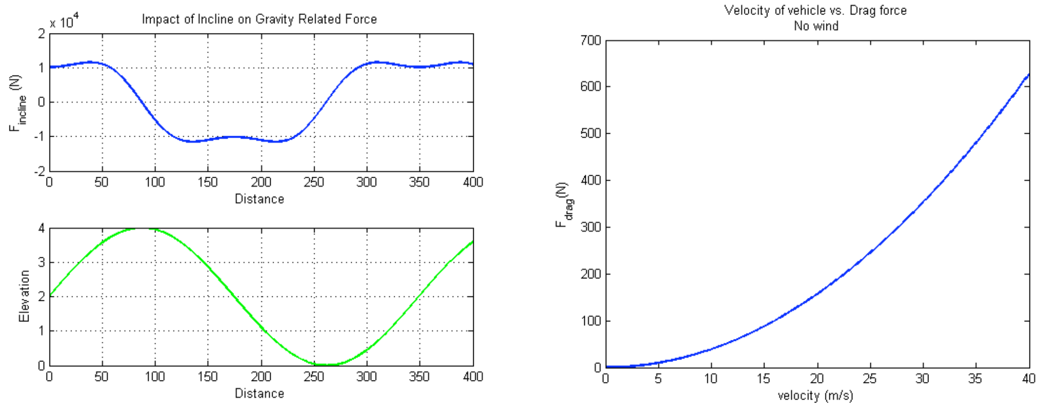

Figure 2.3: **Left**:Relationship between the incline of a hill and the force that acts on the vehicle due to the gravitational pull at that incline **Right**: Relationship between velocity of vehicle and the drag it creates

Unfortunately, this approach is not as simple as it seems. There are several approximations that must be made in this model. The coefficient of drag used in Equation 2.3 is complicated to calculate on a geometry such as a truck. Even something as easy to comprehend as mass in Equation 2.4 is difficult to estimate for a truck that can weigh tons less when empty as compared to when it is full. Equation 2.2 calls for an η, which represents the efficiency of the engine. This value involves several factors including efficiencies of several components in the engine. Different conditions can vary this efficiency. Now that every seemingly simple equation presented has been dissected for inefficiencies, it is safe to say that even a well done model of such a system could never be 100% accurate.

Chapter 3 – Problem Description and Approach

Although predictive cruise control has already had fairly successful results, we believe that applying a learning algorithm to the control system will make the results more accurate in highly variable conditions. A learning environment will account for several variables not easily modeled in traditional control systems. This could result in a greater increase in fuel efficiency.

The problem at hand is that there already exists a model based predictive cruise control algorithm that outputs a desired velocity that is not obtainable by the truck. This problem is demonstrated in Figure 3.1. The red line is the desired velocity, v_{des}, that represents the velocity at which PCC wants the truck travel. The blue line is the actual velocity, v_{act}, that the truck is obtaining.

Rather than learn the entire objective function with the learning controller, we can implement the learning into the system *with* the PCC algorithm. If the the offset between the PCC desired velocity and the actual realized velocity by the vehicle were known, then it would be possible to correct the desired velocity to values that the truck can reach. This would reduce the frequency at which the desired velocity curve needs to be recalculated, thus saving computational time and fuel. Therefore, we want the learner to learn the offset in order to make a correction to the system. This proposed solution is demonstrated in the block diagram in Figure 3.2.

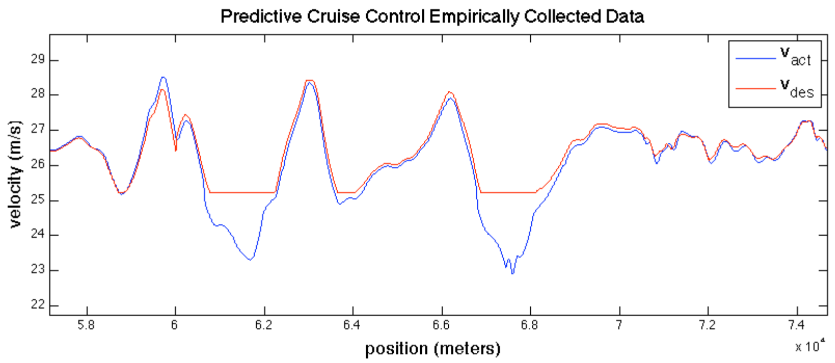

Figure 3.1: PCC error offset: The red line labeled v_{des}, is the desired velocity curve that is outputted from the PCC algorithm. The blue line labeled v_{act}, is the actual velocity that the truck travels. The difference between the two is the *offset*.

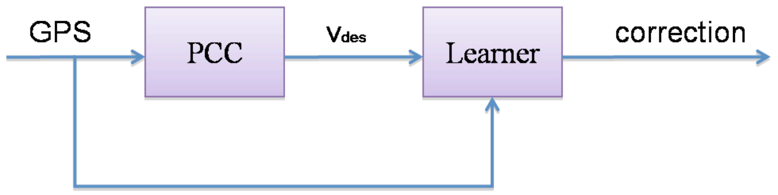

Figure 3.2: Proposed Solution to the PCC problem: Given the model based PCC, we propose to implement a learning based control method in order to predict the offset between the desired velocity from PCC and the actual velocity the vehicle travels. If we know what the error is going to be ahead of time, then we may be able to correct the problem or adjust the desired velocity in order to minimize the number of times PCC needs to recalculate.

As we can see from Figure 3.2, the learner can use information from both the GPS and the desired velocity curve outputted from the PCC. Using that information, it will have to accurately output an offset for each time step. There are several different types of learners that are available for such a task. However, each learner specializes in different areas and it is often difficult to know which one fits the problem at hand. Next, we will discuss some of the different types of learning control methods and why we eventually determined that an Artificial Neural Network approach would be the best fit for this problem.

3.1 Learning Control Methods

There are several learners that could be utilized in this study. The key is to find the learner that is best suited for the problem stated above. Therefore, we will discuss some of the possible learners that could be implemented as well as their pros and cons with respect to this thesis.

3.1.1 Back Propagating Neural Network

Back propagating neural networks use a large data set of input values and target output values to learn unknown functions. This data set is used to “teach” what actions are good, bad, and exactly what to do to make the network better. This learning method is called supervised learning because the data leads the network in the right direction with hard values in order to change the parameters of the network for minimum output error [2, 16, 27, 30, 46].

Pros:

- Can learn any continuous function given the right parameters

Cons:

- Using gradient decent, the solution can get stuck in local minima of objective function
- It is somewhat of an artform to find the best input/output mapping

3.1.2 Neuro Evolutionary Neural Network

Neuro Evolutionary neural networks use unsupervised learning to find the best action. This type of learner has no data examples to follow, but rather takes an action and then receives feedback on its performance. Next, rather than deterministically changing some parameter based on solid data, the network will be randomly changed. The performance of each state will be evaluated and the better will be selected and the worse discarded. This will eventually produce a network that delivers decent performance based on some objective function [1, 8, 21, 27, 30].

Pros:

- No data set is needed
- Less likely to get stuck in local minima due to stochastic search

Cons:

- Slow to get good results due to stochastic search features

3.1.3 Reinforcement Learning

Reinforcement learning is as its title implies. It is a reward based learning controller. That is to say that it learns from its interactions with the environment. After taking an action, the system will receive some feedback from the environment. Based on this feedback, the behavior will be modified until some goal is reached [36, 27, 30, 43].
Pros:

- Applicable to problems with long term vs. short term reward trade off
- Can reward for a sequence of good actions, not just single actions

Cons:

- Slower to converge to solution than back propagation

3.2 Approach: Artificial Neural Networks

Due to the easy access to thousands of sample data points, it was determined that back propagating neural networks would be a good approach for this problem. Throughout the rest of the paper, back propagating neural networks will now be referred to as neural networks or artificial neural networks(ANN).

3.2.1 Algorithm

The neural network algorithm effectively utilizes its extensive data set by learning from experience. A data set consists of input values that correspond to target output values. On the highest level, we can think of a neural network as a black box that takes inputs and produces outputs. When in the learning stage, this output is compared to the target output and an error is calculated. Using this error, changes can be made to the network properties. The next iteration of this sample will produce an output closer to the target value. When we present several samples to the neural network that follow the same basic function, the error will be reduced several times and the parameters changed each time until the error is at a minimum and function is learned.

Figure 3.3 shows the basic structure of a two layer neural network. The inputs can be represented by the neurons in the input layer and the output of the system will be the result of the output layer neurons. In this problem, there will be only one output and so only one output neuron is shown. However, in this paper we will present many different ANN configurations were we will have 4, 6, 8, or 10 inputs.

Although there are only five displayed, the hidden layer can have any number of hidden units. Having a hidden layer allows the ANN to learn more complicated functions. The more hidden units there are, the greater complexity the function can have [30]. Both the hidden and output layer neurons have what is called an activation function. The tangential activation function used for this study takes

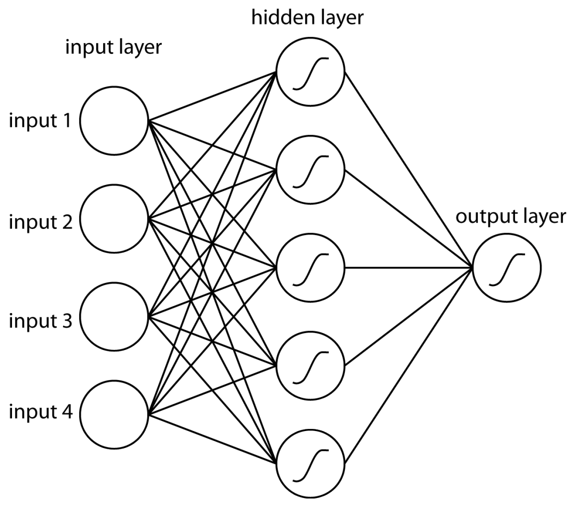

Figure 3.3: A two layer neural network: Simple neural networks can be composed of an input layer, hidden unit layer, and an output layer, with weights connecting every "neuron" in each neighboring layer. The lines drawn are these weights. The number of weight layers gives the neural network its name. This example has two layers of weights separated by the hidden layer, hence it is a two layer neural network. The hidden units and outputs will have some type of activation function. In this study, tangential activation is utilized. The system can have any number of inputs,hidden units, hidden layers, and outputs, but the more there are, the greater the computational time.

the input to the neuron and gives it a value between -1 and 1 that will affect the outputted value. The lines connecting the neurons are extremely important. These lines represent weights. Therefore, every value that is connected from an input neuron to a hidden unit or from a hidden unit to the output will be weighted when fed through the neural network. Figure 3.4 shows the basic structure of a neural network algorithm. The main goal of the algorithm is to update the weights, $v_{i,j}$, and $w_{j,k}$, which correspond to the weights connecting the input layer to the hidden layer and the weights connecting the hidden layer to the output layer respectively.

```
For each epoch, loop:
    For each sample in training set, loop:
        1. Feed forward:
            – Compute hidden layer
              $h_j = tanh(\sum_i x_i v_{i,j} + bias_j)$
            – Compute output layer
              $y_k = tanh(\sum_j h_j v_{j,k} + bias_k)$
            – Compute errors at each output
              $e_k = t_k - y_k$
        2. Back propagate error and calculate weight updates
        3. Update Weights
           new weight= old weight + weight update
           $v_{i,j} = v_{i,j} + \Delta v_{i,j}$
           $w_{j,k} = w_{j,k} + \Delta w_{j,k}$
```

Figure 3.4: Feed Forward Back Propagating Neural Network Algorithm: The neural network algorithm consists of 3 major sections. The first is the feed forward section which uses the input values and weights to ultimately calculate the outputs. The weight $v_{i,j}$ corresponds to the weight between input x_i and hidden unit h_j. The weight $w_{j,k}$ corresponds to the weight between hidden unit h_j and output y_k. There are also bias weights for each hidden unit and each output, which acts to set the threshold for the activation function, *tanh*. To compute the error at each output, the calculated output, y_k is simply subtracted from the target value, t_k. The second step is to back propagate the error through each layer of weights. Using the error as well as a derived form of the activation function and the values at each input and hidden layer, an update for each weight can be calculated. The last step is to update the weights. The next iteration then uses these new weights. Weights are updated until some stopping criteria is met and the most recent weights are then used for the test simulation.

The subscript i, denotes the input, j denotes the hidden unit, and k denotes the output. If there are n inputs, m hidden units, and d outputs, then the v is a $n \times m$ matrix and w is a $j \times k$ matrix. There are also bias weights for each hidden unit and each output, which act to set the threshold for the activation function. These weights are the parameters that can be changed during the learning process. Therefore the function to be learned is F_w, the function as a function of the weights in the neural network [2, 27, 30].

The weights are adjusted to minimize the mean square error at the output level. When the error is back propagated through the layers, it adjust the weights using gradient decent to lower the mean square error. The weights will be adjusted for each sample presented to the network. The network is "trained" in this fashion using what is called a *training set*. The neural network will present every sample in the training set once per epoch. In the next epoch, every sample will be presented again until the error stops improving or some other criteria is met. The weights that are settled on in the last epoch will then be used to test the performance of the system. Testing requires that the system sees a data set that it has never seen before. Therefore, it will be be tested on the *test set* which contains none of the same samples as the training set. Performance is determined by presenting the data set to the network and comparing its simulated output to the desired output and calculating the error between them.

In this paper the full data set is split into a training set and a test set using two different methods. As displayed in Figure 3.5, the first method is a segregated approach where the training set consists of the first third of the data and the test

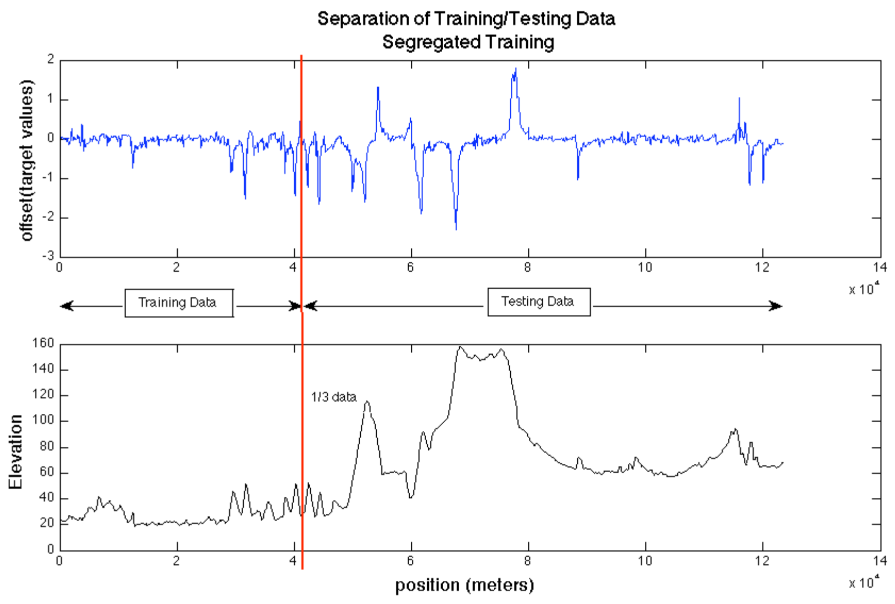

Figure 3.5: Separation of Training and Testing set using segregation: The first method of separating training and testing data is to take the first third of the data for training and the rest will be reserved for testing. This method is very straight forward and easy to implement but may cause large error in the test set when a scenario is presented that is not similar to anything from the training set.

set is the other two thirds of the data. This method is used first because it is all around easy to implement in the code. However, as seen in the figure, the training set does not seem to represent the total data set very well. While we don't want to have the exact data in both the training and test sets, it is important for testing that the system was trained on something *similar*. The second method shown in Figure 3.6 is an integrated approach. One third of the data is still used for training but it is evenly spaced throughout the entire data set so that more scenarios of the

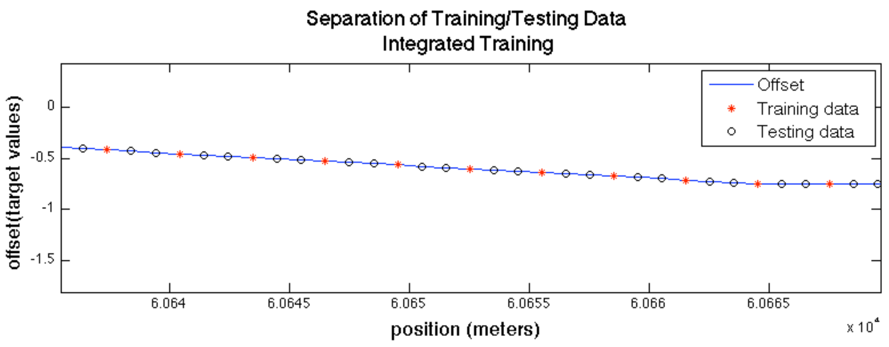

Figure 3.6: Separation of Training and Testing set using an integrated approach: This method of separating the training data will better represent the entire data set since it is inter-dispersed throughout the data. Every third data point is used as a training point and the rest is used for testing.

data are represented. Therefore, every third data point is used for training, and the remaining 2 of 3 points are used in testing.

3.2.2 Input/Output mapping and representation

When an unknown function is presented in a problem such as this one, it is vital to find the most relevant inputs to fit the problem. For example, in this study we are trying to learn the offset between the PCC desired velocity and the actual velocity the vehicle realizes. Factors that would *not* affect this would be things like the trucks cargo, or phase of the moon. Factors that might affect the offset slightly, could be things like temperature of the environment, condition of the road surface, and slight winds. Factors that we suspect play a major role in the offset are the velocity of the vehicle, the PCC desired velocity, and the grade of the incoming terrain. Therefore, simply using the knowledge presented in Section 2.2, the plot we saw in Figure 3.1, and some common sense, we have predicted that information obtained from the PCC desired velocity, and the elevation curve will give us the most relevant inputs to our neural network.

Given this prediction and a prior knowledge of how neural networks perform, we have chosen to train and test the networks based on the 7 different input configurations shown in Figure 3.8. These are all variants of the desired velocity and elevation curves. The left column shows the elevation based input configurations and the right column shows the desired velocity based input configurations. Before we go into greater detail about the different configurations, lets look at what we will be doing with these inputs.

In Figure 3.7 we can see the setup of the learning problem. In this example, the inputs will be from the "elev" configuration. There will be four elevations from equally spaced points within the "look ahead window" (i.e. the furthest distance that the neural network is gathering information from). The "look ahead" distance will help to find how much distance is relevant. For example, we know that the elevation curve contains relevant information; however, an elevation 50 miles ahead of the vehicle will not affect the next 50 meters. In this example, we have a look ahead distance of 100 meters. Therefore, the 4 inputs will be placed at approximately 0, 33, 66, and 100 meters away from the current vehicle location. The offset we are trying to learn will be half of the look ahead distance. In this case, it is 50 meters away. Now that we know some terminology and the basic set up of the learning problem, we can discuss in more detail the plots in Figure 3.8.

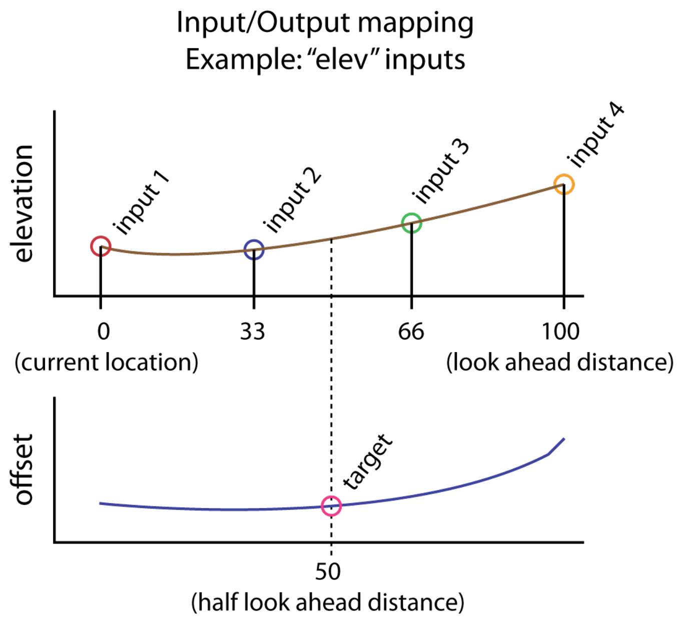

Figure 3.7: Input/Output mapping for a 100 meters look ahead distance using "elev" input configuration: These plots show that the inputs used in the neural network come from evenly spaced points of the elevation or desired velocity curve. The output used for each sample is the offset at 1/2 the look ahead distance. In this example, the look ahead distance is 100 meters, therefore the inputs will be the elevation at a distance of 0, 33, 66, and 100 meters, while the output will be the offset at a distance of 50m.

Starting on the top left and going down, we see that the "elev" configuration inputs elevations at equally spaced points within the look ahead window. The "grade" configuration inputs the road grade of the road at these same points. In the "slopes" and "bridge" configurations, we have to set up one more equally spaced point in the look ahead window in order to get the same number of inputs. The inputs for "slopes" are the slopes of the lines that connect the current elevation to the four future elevations. The inputs for "bridge" are the slopes between

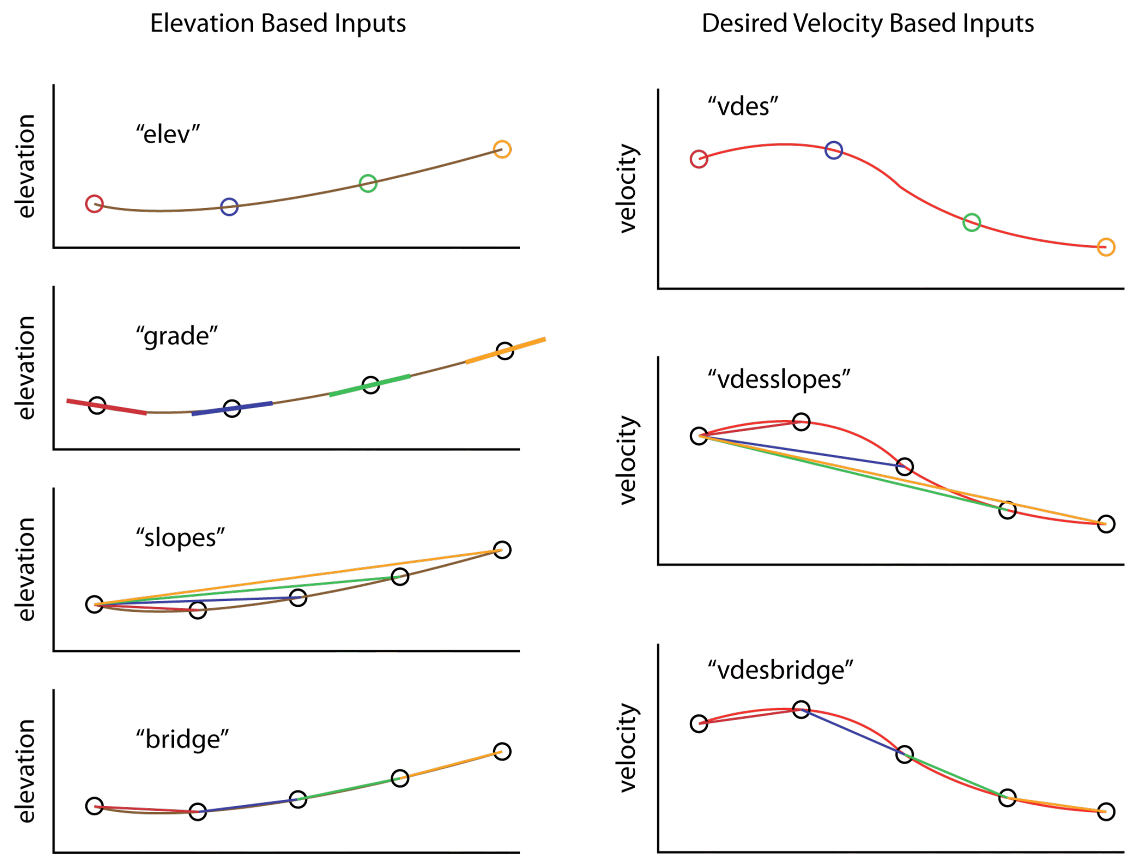

Figure 3.8: Configurations for neural network inputs: The neural network is exposed to several different configurations in order to see what information is most relevant to our desired target. Configuration **"elev"** inputs elevation values at equally spaced points within the look ahead window. Configuration **"grade"**inputs the road grade at equally spaced points within the look ahead window. Configuration **"slopes"** inputs the slopes from the current elevation to n evenly spaced elevations in the look ahead window. For this example, n, the number of inputs, is 4. Configuration **"bridge"** inputs the slopes between each neighboring equally spaced elevation, like the slopes of a connect-the-dots puzzle. Configuration **"vdes"** inputs the desired velocity values at equally spaced points within the look ahead window. Configurations **"vdesslopes"** and **"vdesbridge"** are calculated in the same way as their elevation based counter parts, except the desired velocity curve is used to extract the input values.

each neighboring elevation point, like the slopes of a connect-the-dots puzzle. On the right column, the “vdes” configuration inputs the four equally spaced desired velocities within the look ahead window. Do not confuse the “vdes” configuration with the desired velocity curve, v_{des} which is the actual curve while “vdes” is referring to the input configuration mapping. The “vdesslopes” and “vdesbridge” configurations follow the same rules as “slopes” and “bridge” respectively, however they use the desired velocity curve instead of the elevation curve. Note that the desired velocity based inputs underwent a preprocess smoothing procedure. In the data, the PCC model based control system recalculates the desired velocity every 4000 meters. This results in discontinuities in the data where it will abruptly change to meet the actual velocity curve. Since neural networks cannot learn discontinuous functions, it was vital to smooth this data out. To smooth the data, each point in the vdes curve is replaced with an average of the values of itself and the nearest 20 points.

The reason these configuration variants of the raw data were created was to see which form has the relevant information most readily available to the neural network. Choosing inputs is somewhat of an art. At times, trying different variants of the same information is the only way to get the best results possible.

Matlab’s neural network toolbox aided this study. Therefore, many of the neural network parameters were determined by the toolbox. Several conditionals including gradient, validation checks, and a maximum number of 1000, were used to determine the number of epochs. Learning rate was optimized within the toolbox. Number of hidden units was user modified to be 20 hidden units for the 4 and 6 input variations and 40 hidden units for the 8 and 10 input variations. The reason for this parameter set is that greater numbers of inputs will require a greater number of hidden units to learn the function.

A common approach to Neural Networks is to move from simple inputs to more complex. Starting with the easiest information to process and implement in the algorithm is always a good idea. Complexity can then be added as needed to improve the performance of the learner. As we discussed earlier, the greater the number of neurons, the more computational time is required. Therefore, we have started with 4 inputs with a look ahead distance of 100 meters in our examples but will also see the affects of 6, 8, and 10 inputs as well as look ahead distances of 400 and 800 meters. We have also started by using one type of input for each configuration. That is, the inputs are in the same form and derive from either the desired velocity curve or the elevation curve. The results for these "single-concept" neural networks will be presented in Chapter 4. In Chapter 5 we will see the effects of using multi concept inputs. That is, we will enrich the input space by using both elevation based and desired velocity based inputs in the neural network to see if the addition of information is beneficial to the system.

Chapter 4 – Single Concept Inputs

The first results we will discuss come from the single concept input configurations that were presented in Figure 3.8. Over three hundred different single concept variations of neural networks were run to find out which inputs and training methods would perform the best. The varied parameters included:

1. **Configuration:** Seven configurations as described in Section 3.2.2 and Figure 3.8 were used. Representing the data in different ways will give us an idea of what aspects of the environment affect the PCC error the most.

2. **Number of Inputs:** The number of inputs were varied to see how many inputs were needed to sufficiently represent the system. Fewer inputs are desirable because greater number of inputs will be computationally taxing. We experimented with 4, 6, 8, and 10 inputs.

3. **Look Ahead Distance:** How much of the future terrain affects the offset? If we don't look far enough then we are not getting enough information, but looking too far will result in extraneous data. For this study, 100, 400, and 800 meter look ahead distances were sampled.

4. **Training Method:** Two training methods were used: Segregated and Integrated training as demonstrated in Figure 3.5 and Figure 3.6 respectively.

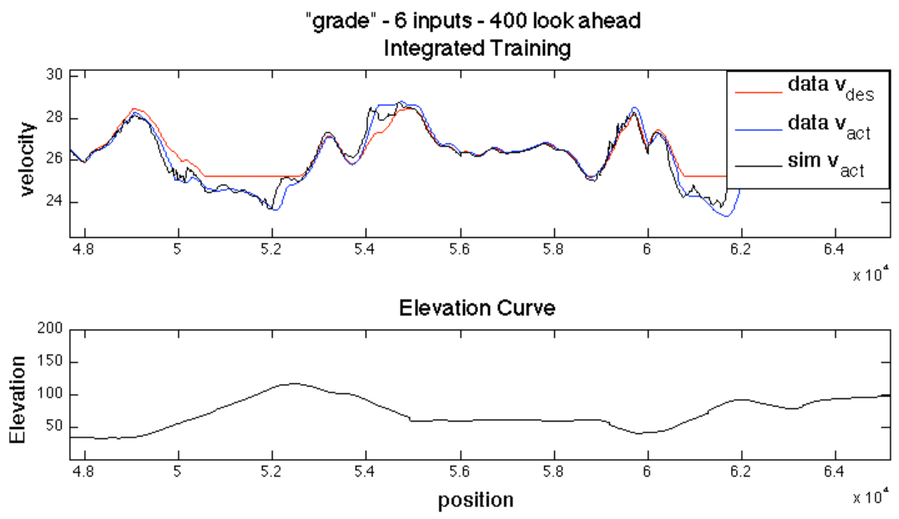

Figure 4.1: **top**: Results from one run with “grade” configuration, 6 inputs, 400 look ahead window, using integrated training **bottom**: The elevation curve corresponding to the position of the simulation results.

5. **“No limits” analysis:** “No limits” analysis is another variation of training and testing that has not yet been introduced. A brief description of this analysis is that it only uses a subset of the data. Results for “no limits” analysis will be presented in Section 4.4.

The results will be presented in two main forms: Graphical and tabular. Using both of these methods together will give a much better idea of how the learners are performing than if we only used one or the other.

Plotted in Figure 4.1(top) are three curves. The first blue curve is the actual velocity(data v_{act}) of the vehicle and the red line is the PCC desired velocity(data v_{des}) curve. The black line represents the actual velocity that the neural network has simulated as a prediction(sim v_{act}). Therefore, we want the black line to be as close as possible to the blue line for best results. At worst, we want the black line to be closer to the blue line than the red line is so that at least our learning controller improves upon the model based PCC. Figure 4.1(bottom) is the elevation curve that corresponds to the positions on the top plot. The elevation curve is plotted beneath the results because the simulation used “grade” inputs which utilizes information from elevation. Visually seeing where inputs come from can help one to understand why a learner is doing what it does.

Table 4.1: 400 look ahead- segregated training All test terrain

Configuration	4inputs	6inputs	8inputs	10inputs
no learning	0.225			
elev	$0.719/-219\%$	$1.001/-344\%$	$0.538/-138\%$	$0.543/-141\%$
grade	0.165/26.7%	0.165/26.5%	0.190/15.5%	0.203/9.6%
slopes	0.173/23.0%	0.187/17.1%	$0.241/-6.9\%$	0.223/1.8%
bridge	0.181/19.8%	0.179/20.5%	$0.234/-4.1\%$	0.216/4.2%
vdes	0.167/25.6%	0.171/24.1%	0.217/3.4%	0.199/11.6%
vdesslopes	0.208/7.4%	0.204/9.4%	$0.243/-8.1\%$	$0.240/-6.7\%$
vdesbridge	0.205/8.8%	0.210/6.5%	$0.239/-6.0\%$	0.216/4.2%

Such plots as seen in Figure 4.1 are very interesting as a visual. However, after looking at the hundreds of result plots that can come from the hundreds of variations itemized earlier in this chapter, all these plots start to look very similar and the actual performance cannot be ascertained. Therefore, it is important to know quantifiable measures of how the learners are performing. Table 4.1 shows two things. Each cell represents the results of a single run, training and testing, through the neural network using the corresponding configuration(rows) and number of inputs(columns). Therefore, each table will display 28 different runs(7 configurations $\times$ 4 inputs). During each run, every testing point simulates a prediction of what the actual velocity will be. The offset between the simulated v_{act} and the data v_{act} is calculated for every testing point and averaged to find the first value in each cell. The offset distance without learning is simply the difference at every point between the desired velocity and the actual velocity. These offsets will have units of meters/second since they are velocities. Similar to how we wanted the simulated v_{act} (black line) to be closer to the data's v_{act} (blue line) than the PCC's desired velocity, v_{des} (red line), we want the learners offset to be smaller than PCC's offset.

The second value in the cell represents the percent improvement that the learner has obtained over the "no learning" scenario. For Table 4.1, the "no learning" scenario resulted in an average of 0.225 meters/second over the entire testing set. Any value lower than that means that the learner has improved the system by some positive percentage. Any value greater than that means that the learner made the system worse and will result in a negative percent improvement.

The single concept results chapter will present how changing different parameters of the neural network can affect the overall outcome. First we will discuss the benefits and limitations of post process smoothing. Secondly, the results will be dissected into different terrain classifications, and we will see how the systems perform on flat, uphill, and downhill road segments. Next we will see how number of inputs and look ahead distances impact the results. Lastly, we will present "no limits" analysis and how well the neural networks performed under these new conditions.

4.1 Post Process Smoothing

To generate the best results possible we first post processed the data. The nature of this neural network is that every input is slightly different and the previous predicted error from the iteration before is not known. The product is sometimes bumpy results. If we were to implement these values into the vehicle control system, the vehicle would have difficulties following the path. Due to the nature of the neural network used in this study, post process smoothing is a very beneficial tool.

Figure 4.2(top) presents the raw results from the "grade" configuration using 4 inputs, a 100 look ahead distance, and segregated training. In Figure 4.2(bottom), we see the same plot but smoothed using an average of the nearest surrounding 100 data points. We can see that the results look much better and appear to fit more closely to the target values.

Table 4.2 shows quantitatively how the results from the learner can be improved upon with smoothing techniques. The first of the three table series shows the raw results from all configurations and all number of inputs from the 400 look ahead, segregated training scenario. The second table gives the results from the same runs that have been smoothed using 100 data points. The third table is another representation of the same data being smoothed with 200 data points. In this case, greater smoothing produces better results. This is not to say that more smoothing will always produce more benefit. Smoothing with maximum amount of data points would end up in a straight line at the average velocity. Another

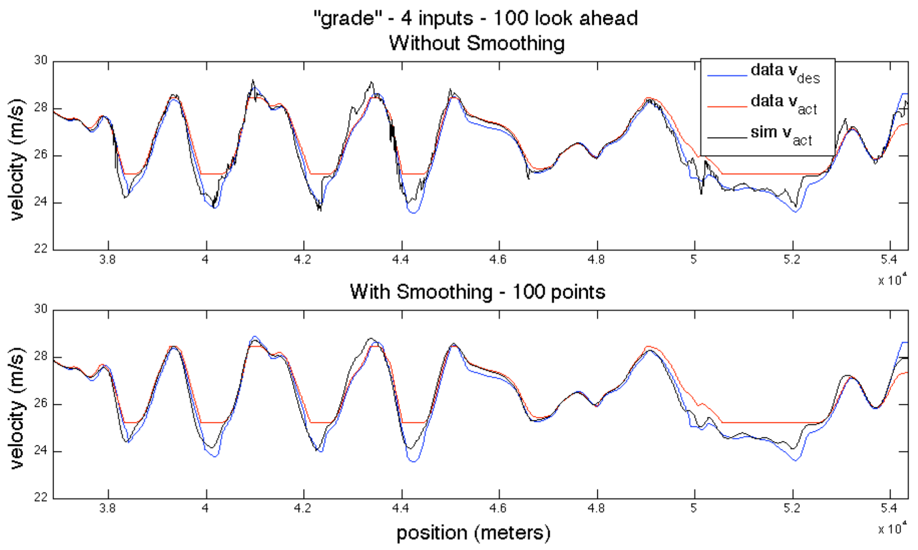

Figure 4.2: **top**: Results without smoothing **bottom**: Results with smoothing - Post process smoothing helps to obtain better improvement in the system as well as a velocity curve that could realistically be followed by a vehicle.

thought to consider is that large amounts of smoothing may benefit the overall performance, but may hurt sections of the results. For example, smoothing may help in hilly terrain and decrease the improvement in flat terrain resulting in an overall improvement, but not a desirable result. This is just to show that smoothing does help in moderation. Smoothing with too many points will result in greater error. It is important to find a good balance that optimizes the benefit of this process.

Table 4.2: Smoothing benefits: 400 look ahead- segregated training

Raw results - not smoothed

Configuration	4inputs	6inputs	8inputs	10inputs
no learning	0.225			
elev	0.719/ − 219%	1.001/ − 344%	0.538/ − 138%	0.543/ − 141%
grade	0.165/26.7%	0.165/26.5%	0.190/15.5%	0.203/9.6%
slopes	0.173/23.0%	0.187/17.1%	0.241/ − 6.9%	0.223/1.8%
bridge	0.181/19.8%	0.179/20.5%	0.234/ − 4.1%	0.216/4.2%
vdes	0.167/25.6%	0.171/24.1%	0.217/3.4%	0.199/11.6%
vdesslopes	0.208/7.4%	0.204/9.4%	0.243/ − 8.1%	0.240/ − 6.7%
vdesbridge	0.205/8.8%	0.210/6.5%	0.239/ − 6.0%	0.216/4.2%

Smoothed with 100 surrounding data points

elev	0.717/ − 218%	0.996/ − 342%	0.531/ − 136%	0.538/ − 139%
grade	0.161/28.5%	0.161/28.3%	0.180/19.9%	0.194/13.8%
slopes	0.164/26.9%	0.178/21.1%	0.221/1.8%	0.202/10.4%
bridge	0.171/24.0%	0.168/25.2%	0.210/6.5%	0.190/15.5%
vdes	0.164/27.2%	0.167/25.8%	0.202/10.1%	0.188/16.4%
vdesslopes	0.200/11.3%	0.194/13.6%	0.226/ − 0.5%	0.220/2.2%
vdesbridge	0.197/12.4%	0.198/12.0%	0.221/1.9%	0.201/10.8%

Smoothed with 200 surrounding data points

elev	0.715/ − 217%	0.989/ − 339%	0.526/ − 133%	0.535/ − 137%
grade	0.159/29.1%	0.161/28.5%	0.174/22.5%	0.189/16.1%
slopes	0.160/28.8%	0.175/22.4%	0.214/4.7%	0.195/13.5%
bridge	0.167/25.7%	0.164/27.0%	0.199/11.8%	0.179/20.3%
vdes	0.161/28.3%	0.166/26.5%	0.191/15.2%	0.180/19.9%
vdesslopes	0.195/13.3%	0.192/14.6%	0.218/2.9%	0.214/4.8%
vdesbridge	0.194/14.0%	0.194/13.9%	0.214/4.9%	0.197/12.5%

4.2 Terrain Results

Next, we are interested is seeing where in the test set improvements (or degenerations) are being made. The total improvement over the entire terrain will be presented as well as a further dissection that shows how the network improves over different areas of the data set. The terrain of the testing data was classified into either “flat”, “uphill”, or “downhill”. The terrain was parsed using grade as the classifier. If the average of the grades within the look ahead window was greater than or equal to .015, then that area was classified as “uphill”. Where as if the average grade was less than or equal to -.015, then the area was classified as “downhill”. All points with an average grade between -.003 and 003, was considered “flat”. Figure 4.3 demonstrates the areas and their labels.

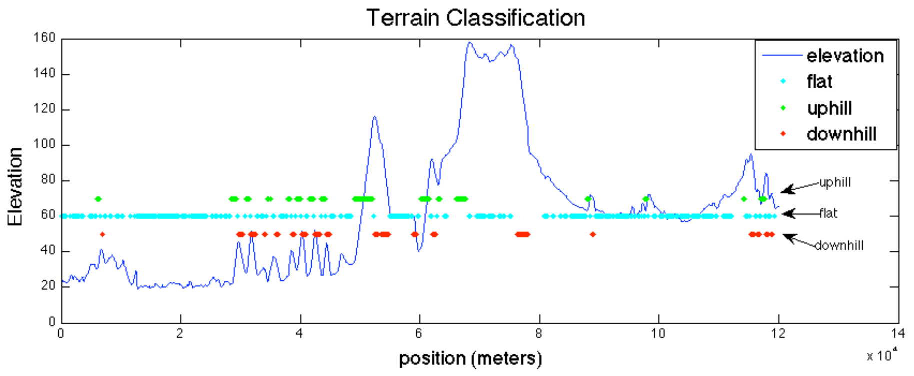

Figure 4.3: Terrain classification: The value of the grade in each area was used to determine what sections of the terrain would be classified as flat, uphill, and downhill. Other classifications included “midranged” and was not used in the analysis.

Table 4.3: All test terrain with 100 look ahead - segregated training - smoothed with 200 pts

All Terrain - smoothed with 200 pts

Configuration	4inputs	6inputs	8inputs	10inputs
no learning	0.225			
elev	1.270/ − 465%	1.193/ − 430%	1.291/ − 474%	0.440/ − 95.6%
grade	0.170/24.1%	0.165/26.6%	0.172/23.3%	0.172/23.5%
slopes	0.585/ − 160.2%	0.162/28.1%	0.585/ − 160.3%	0.162/27.9%
bridge	0.168/25.4%	0.157/30.3%	0.164/26.9%	0.168/25.4%
vdes	0.172/23.6%	0.165/26.5%	0.181/19.3%	0.183/18.8%
vdesslopes	0.585/ − 160.3%	0.190/15.3%	0.194/13.8%	0.194/13.7%
vdesbridge	0.190/15.3%	0.194/13.8%	0.195/13.2%	0.196/12.8%

Flat Terrain - smoothed with 200 points

no learning	0.093			
elev	1.360/ − 1356%	1.266/ − 1255%	1.373/ − 1370%	0.409/ − 338%
grade	0.084/10.4%	0.085/9.2%	0.086/7.5%	0.086/7.7%
slopes	0.492/ − 427%	0.085/9.2%	0.492/ − 427%	0.086/8.3%
bridge	0.088/5.9%	0.085/9.3%	0.086/7.9%	0.088/5.8%
vdes	0.090/3.5%	0.087/6.7%	0.092/1.2%	0.089/4.89%
vdesslopes	0.492/ − 427%	0.104/ − 10.9%	0.107/ − 14.7%	0.105/ − 12.1%
vdesbridge	0.101/ − 11.4%	0.103/ − 10.2%	0.102/ − 9.4%	0.106/ − 13.8%

Uphill Terrain - smoothed with 200 points

no learning	0.765			
elev	0.672/12.1%	0.584/23.6%	0.697/8.9%	0.510/33.3%
grade	0.458/40.1%	0.404/47.2%	0.429/43.9%	0.399/47.8%
slopes	1.209/ − 58.2%	0.348/54.5%	1.209/ − 58.2%	0.366/52.1%
bridge	0.390/49.0%	0.324/57.6%	0.367/51.9%	0.362/52.7%
vdes	0.363/52.6%	0.350/54.2%	0.409/46.5%	0.411/46.2%
vdesslopes	1.209/ − 58.2%	0.443/42.1%	0.442/42.2%	0.437/42.9%
vdesbridge	0.437/42.9%	0.445/41.8%	0.470/38.6%	0.446/41.6%

Downhill Terrain - smoothed with 200 points

no learning	0.435			
elev	1.460/ − 235%	1.498/ − 244%	1.523/ − 249%	0.617/ − 41.7%
grade	0.408/6.3%	0.416/4.4%	0.442/ − 1.5%	0.471/ − 8.3%
slopes	0.504/ − 15.7%	0.439/ − 0.8%	0.505/ − 16.0%	0.418/4.0%
bridge	0.425/2.3%	0.423/2.8%	0.432/0.8%	0.446/ − 2.4%
vdes	0.418/3.9%	0.415/4.7%	0.436/ − 0.2%	0.454/ − 4.3%
vdesslopes	0.505/ − 16.0%	0.458/ − 5.2%	0.469/ − 7.8%	0.474/ − 8.9%
vdesbridge	0.465/ − 6.7%	0.470/ − 8.0%	0.462/ − 6.2%	0.470/ − 8.0%

Table 4.3 shows the performance over the entire data set, the flat areas ,the uphill areas, and the downhill areas. All these tables come from the same 28 runs that were produced using a 100 meter look ahead distance, segregated training and smoothed with 200 surrounding data points.

From these tables we observe that most of the improvement is made on the uphill portions of the data while the improvement on the flat and downhill areas are less and sometimes are not improving at all, but worsening. The improvement distribution is probably because the PCC desired velocity achieves better offset values on the flat and downhill portions and very bad offset values on the uphill terrain. Therefore, there is more room for improvement when PCC is failing. These trends are seen throughout all the variations.

A side observation that does not pertain to terrain trends, involves the exceedingly poor performance of the "slopes" configuration with 4 and 8 inputs, the "vdesslopes" configuration with 4 inputs, and all of the "elev" configurations. These large negative improvements can be seen all the way through each terrain table. These values can be explained.

The "elev" configuration performs very poorly when using segregated training because the inputs are the actual elevation values. Looking at the segregated training figure, Figure 3.5, we notice that the elevations in the second half of the terrain sees much greater heights. Since the neural network has not seen such values in the training examples, it doesn't know what to do during the test set. However, in the integrated training scenario as seen in Table 4.4, the "elev" configuration does very well because it has seen these values before. This implies that the neural network has put great importance on the “elev” input *values* rather than the relationship between the inputs(i.e. the change in elevation or grade). Hence, even if “elev” outperforms “grade” using integrated training, “grade” would still be a better choice of configuration because it will apply to any road in the world, where as to get a good result for “elev”, training examples at all possible ranges of elevations would have to be gathered and implemented at high cost.

Table 4.4: All test terrain with 100 look ahead - integrated training - smoothed with 200 pts

Configuration	4inputs	6inputs	8inputs	10inputs
no learning	0.195			
elev	0.125/36.1%	0.125/36.1%	0.113/42.3%	0.116/40.4%
grade	0.140/28.3%	0.136/30.2%	0.136/31.6%	0.131/32.9%

The configurations “slopes” and “vdesslopes” have done poorly for a different reason. Recall that each cell in the table comes from a single run. While most runs perform well, others can fail. These two configurations have demonstrated inconsistency in their learning ability. Since back propagating neural networks use gradient decent to converge to the the minimum training error, it is possible to get stuck in local minima in the objective function. When this happens, a less than ideal solution can be produced. However, if the training is closely monitored, these configurations could still be practically implemented.

4.3 Parameter Impact on performance

Parameters including number of inputs, look ahead distance, configuration, and training method define the system and ultimately determine how the learners will perform. We will now try to find the trends that form from all the parameter variations.

The results from all the raw data(not smoothed) have been compiled and displayed in Table 4.5. In this table we can compare the results between differing training methods as well as number of inputs, look ahead distance, and configurations. Given the versatility of this table, it will be referenced in other sections to come. Please note that for space efficiency, only the percent improvement is shown. Raw data was chosen over smoothed for comparison purposes. Smoothing as a post process to this results could improve them by up to 9 percent units. All results that exceed 20% improvement will be bolded for quick reference.

We will specifically discuss the impact of input quantity and look ahead distance in Section 4.3.1 and Section 4.3.2 respectively. Trends that arise in the configuration and training method will be mentioned as they present themselves.

4.3.1 Number of Inputs

All the tables we have seen so far are compiled from single runs. In some cases it is misleading to detect trends when only looking at single runs. To remedy this, 4 configurations were selected and ran 30 times at each of the 4 different input quantities. This gives some amount of statistical significance and will enable us to

Table 4.5: All raw results both segregated and integrated training

100 look ahead window

	Segregated Training				Integrated Training			
Configuration	4inputs	6inputs	8inputs	10inputs	4inputs	6inputs	8inputs	10inputs
no learning	0.225				0.195			
elev	−466%	−432%	−477%	−103%	**37.8%**	**38.5%**	**43.9%**	**41.8%**
grade	19.8%	**23.0%**	17.0%	16.7%	**27.2%**	**29.2%**	**30.0%**	**31.6%**
slopes	−160%	**23.1%**	−160%	**20.1%**	**24.7%**	**26.3%**	**25.8%**	**23.8%**
bridge	**20.9%**	**24.5%**	19.5%	16.8%	**24.0%**	**25.1%**	**27.6%**	17.3%
vdes	**21.5%**	**25.0%**	9.9%	9.2%	**23.0%**	**23.1%**	**25.2%**	**24.5%**
vdesslopes	−160%	10.8%	7.9%	7.1%	15.5%	15.8%	14.6%	12.0%
vdesbridge	11.5%	8.9%	6.9%	5.2%	−866%	9.4%	5.4%	2.5%

400 look ahead window

	Segregated Training				Integrated Training			
Configuration	4inputs	6inputs	8inputs	10inputs	4inputs	6inputs	8inputs	10inputs
no learning	0.225				0.195			
elev	−219%	−344%	−138%	−141%	**25.4%**	**24.7%**	**32.7%**	**32.4%**
grade	**26.7%**	**26.5%**	15.5%	9.6%	17.9%	14.3%	**23.4%**	**23.6%**
slopes	**23.0%**	17.1%	−6.9%	1.8%	15.9%	14.8%	**21.8%**	**23.8%**
bridge	19.8%	**20.5%**	−4.1%	4.2%	17.1%	16.0%	**22.6%**	**23.5%**
vdes	**25.6%**	**24.1%**	3.4%	11.6%	17.2%	15.9%	**22.2%**	**22.3%**
vdesslopes	7.4%	9.4%	−8.2%	−6.7%	−1.1%	6.0%	5.3%	6.7%
vdesbridge	8.8%	6.5%	−6.0%	4.2%	7.6%	4.0%	−1.9%	5.1%

800 look ahead window

	Segregated Training				Integrated Training			
Configuration	4inputs	6inputs	8inputs	10inputs	4inputs	6inputs	8inputs	10inputs
no learning	0.226				0.195			
elev	−440%	−140%	−137%	−138%	9.2%	−866%	11.5%	11.8%
grade	−0.2%	−12.2%	−10.1%	−7.7%	3.3%	1.8%	7.3%	7.7%
slopes	−1.8%	−1.4%	−32.6%	−26.2%	2.8%	5.0%	6.7%	6.6%
bridge	−7.2%	−4.5%	−25.7%	−41.3%	1.5%	4.2%	6.2%	7.0%
vdes	−1.4%	−37.3%	−45.7%	−24.8%	0.5%	3.4%	5.7%	6.5%
vdesslopes	−14.1%	−11.8%	−29.1%	−31.5%	−9.1%	−9.6%	−10.6%	−10.3%
vdesbridge	−159%	−16.8%	−24.8%	−35.0%	−10.3%	−14.6%	−11.6%	−11.9%

see how the number of inputs affect the overall performance of the learner. It will also let us compare the performances of the different configurations.

Figure 4.4 shows the results of this test. The learner implemented used 400 look ahead data, segregated training, and 4 configurations: “vdes”, “grade”, “slopes”, and “vdesslopes”. These configurations were chosen so as to have two from the elevation based inputs and two from the desired velocity based inputs. The number of inputs is represented on the x-axis of the plot. Percent improvement based off the PCC desired velocity offset is the quantity being evaluated on the y-axis. The error bars are calculated using the standard deviation(σ) and the number of runs(N), such that the length of the bar is $2E$. Where $E = \frac{\sigma}{\sqrt{N}}$.

Note that to save on computational time for the statistical plots, the maximum number of epochs trained was decreased to 250 epochs from the original 1000 maximum that was used for the single runs.

First, the unusual data points will be addressed. The “vdesslopes” configuration with 8 inputs and the “slopes” configuration with 6 inputs both have large error bars compared to the rest. Also, these points do not follow the same trend as the others. This likely goes back to the explanation that these configurations are inconsistent. In a set of 30 data points, it only takes one or two outliers to taint the average and standard deviation.

The “vdesslopes” data point at 10 inputs doesn’t have a large error bar but is still in the negative improvement range. This means that this particular variation plainly did not do well. The tables we have seen so far verify that “vdesslopes” and “vdesbridge” have generally done worse than the other configurations.

The trends that we see in the rest of the data show comparable improvements for 4 and 6 inputs, but then a decline in improvement when there are 8 or 10 inputs.

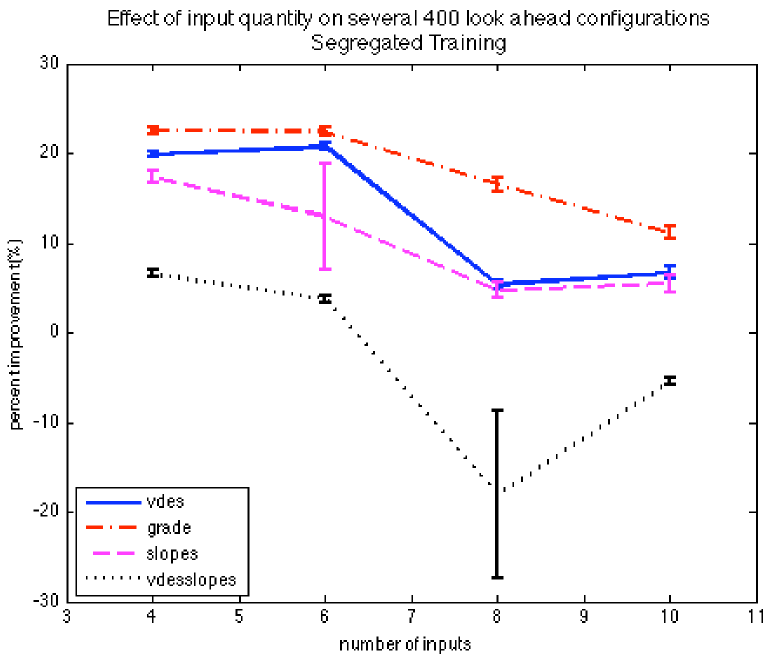

Figure 4.4: Impact of input quantity on percent improvement: Different configurations, all with a look ahead distance of 400 meters, were ran through the neural network 30 times with 4, 6, 8, and 10 inputs.

This is counter intuitive because one would think that more inputs would represent the system better or at least equally. This result could mean that greater inputs are not necessary and therefore give extraneous information. The fault could also lie with the neural network set up. Twenty hidden units were used for the 4 and 6 input variants while 40 hidden units were used for the 8 and 10 input variants. However, all of the networks were subject to a maximum of 250 epochs. It might be that the many more weights that a 40 hidden unit network has to learn were not learned as well within that epoch restriction. However, if it had more time to learn, would it have performed better than or on par with the fewer input examples? Both the 4 and 6 inputs resulted in similar improvements which leads one to believe that more inputs would not improve the system.

We can check this hypothesis by looking at the training error plots that track the mean square error of the system at each training epoch. We will compare the training error plots for both the 6 input and 10 input variants of the "vdes" configuration.

Figure 4.5 shows the comparison of the two plots using the average and standard deviation for the 30 training runs. These plots show that the 10 input variant had enough time to converge to an error. The 10 input error even converges to a lesser value than the 6 input scenario. This leads one to think that there then could be another flaw in the system. Forty hidden units could be overtraining the network, making it learn the training data *too* well and resulting in poor testing results. Another supporting fact that this outcome may be due to overtraining, is that integrated training generally gets better results as input number increases as seen

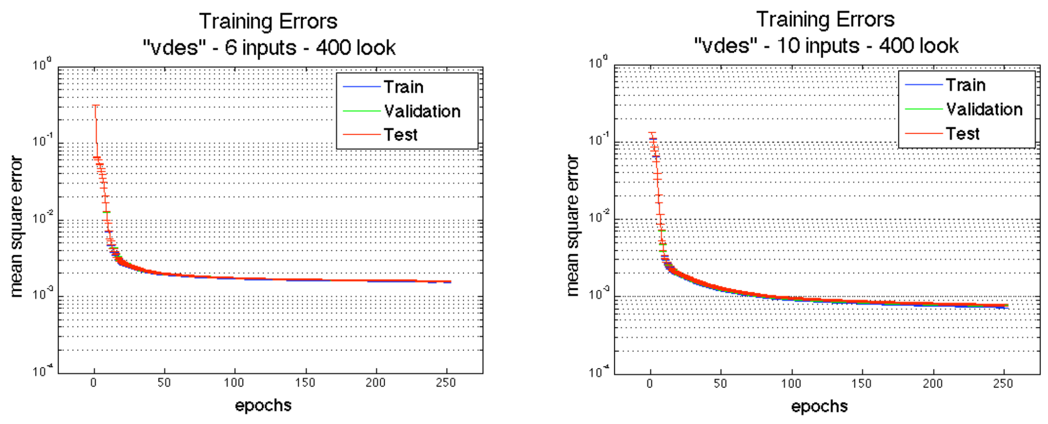

Figure 4.5: **Left**:Training error for the 6 input variant **Right**: Training error for the 10 input variant. Plots consist of training, validation, and test errors. Validation sets are used in training as part of the epoch stop criteria to prevent overtraining.

in Table 4.5. Since the segregated training uses data that is unlike the testing data, overtraining is detrimental, where as the integrated training uses data that is almost exactly like the testing data and therefore overtraining could actually be good for the testing results.

Coming to the conclusion that the 8 and 10 input variations are overtrained, means that we would only want to use this system for a practical application if there is access to training data that is highly representative of the test set. The benefit of segregated training is that we are able to see how the system would perform given a whole new environment.

If the 8 and 10 input variations were not overtrained with too many hidden units, it is suspected that they still would not have done much better than their lesser input versions. The reasoning behind this prediction is that the 4 and 6 input variations were not overtrained and the 6 inputs performed about the same as 4 inputs.

4.3.2 Look Ahead Distance

This section will specifically address the impact of look ahead distance on the results as well as further discuss configuration and training method performance.

Using the same configurations as Figure 4.4, a 30 run statistical investigation into the impact of look ahead distance was conducted using 100, 400, and 800 meter look ahead distances. Number of inputs was held constant at 4. Once again, the number of epochs per run was capped at 250. Figure 4.6 displays the results of this experiment.

The "vdesslopes" at 4 inputs has once again demonstrated this configurations inconsistency. This data point will be excluded from the trend analysis. Although it was initially thought from the single run analysis that 400 look ahead distance did better than the 100 look ahead, the trend in the figure clearly shows that increasing the look ahead distance, decreases the improvement of the system. It is confirmed by Table 4.5, that a look ahead distance of 800 meters will result in drastically worse results. Integrated training results from the same table verify the same trend.

The results shown are conclusive. A look ahead distance of 400 meters or greater presents the neural network with extraneous information that is not needed to predict the offset and will even hurt the system performance.

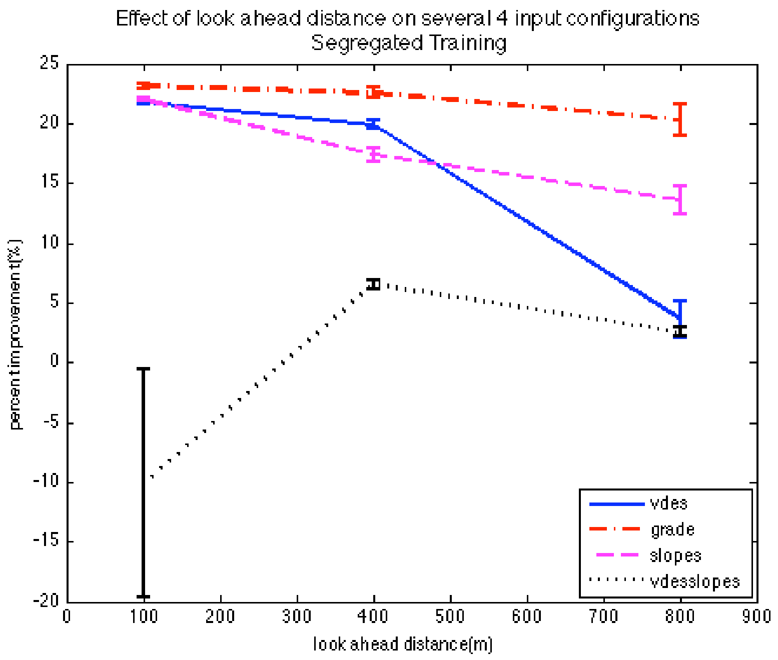

Figure 4.6: Impact of look ahead distance on percent improvement: Different configurations, all with 4 inputs, were ran through the neural network 30 times with 100, 400, and 800 meter look ahead distances.

Consolidating the information gained from Section 4.3, we can make several observations about configuration performance. The most obvious observation is that the “grade” configuration generally produces the largest improvements. “vdes” performs the best of the desired velocity based inputs because “vdesslopes” and “vdesbridge” typically do the worse of all the other configurations. Similarly, “slopes” and “bridge” maintain about the same improvement values. The most complicated configuration is “elev” which does terribly with segregated training and very well with integrated training. On that same note, we have also considered that an input mapping that performs well using segregated training is preferred over one that only does well on integrated training because that implies it adapts better to new terrain and is therefore a more robust mapping.

4.4 Removing limit areas

There was interest to see how this system would behave when certain areas of the data were excluded from the training and testing procedure. The areas where the desired velocity curve reaches its minimum allowable speed and flattens out are called lower velocity limits. Much less of an issue are the upper velocity limits where the vehicle has reached its maximum allowable speed. The largest offsets between the PCC desired velocity and the actual velocity occur at the lower velocity limits. These are also the areas where our learners have been making the largest improvements to the PCC system.

The questions posed in “no limits” analysis is:

How well would the learner perform if all the areas where the desired velocity has reached its lower velocity limit were removed from the data set?

Figure 4.7 displays several areas where the desired velocity has reached its lower limit. The red stars represent each position that corresponds to a data point that will be excluded from the training and testing set. Data starts to be excluded before it reaches the limit area due to the look ahead window. The larger the look ahead distance is, the more data will be excluded.

During a “no limits” scenario, it is likely that the vehicle has reached a long incline in the road and is called to decelerate until it reaches the minimum allowed velocity and then hold at that speed. However, the truck is so massive that it is difficult to simply stop decelerating, so it overshoots the target speed. This overshoot cannot be prevented and therefore learning the offset will not lead to better performance on a practical level. This leads to the “no limits” rationale that we might as well focus on improving the performance of the other areas more effectively. These other areas are mostly flat, mid-range, and downhill terrain.

The raw results from every test is presented in Table 4.6. From this table there are several observations that are immediately noticeable:

- Most of the segregated data produces negative improvement which means the function was not learned at all.
- Most of the integrated data produces positive improvement.

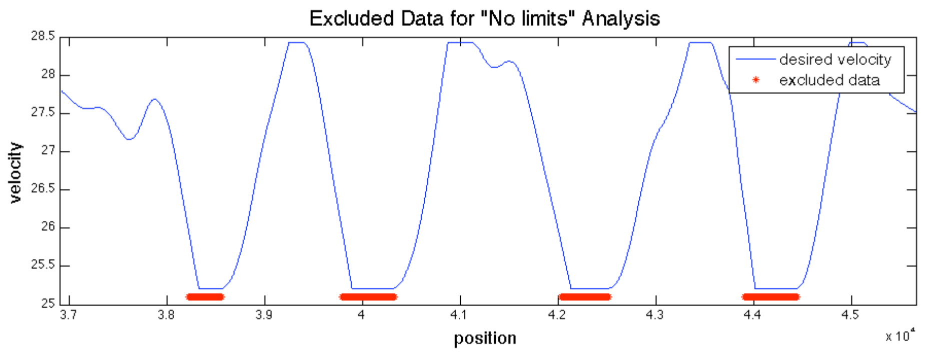

Figure 4.7: Excluded data from “no limits” analysis: All data involving a desired velocity that has reached its lower velocity limits will be excluded from the training and testing samples.

- Segregated training data yields worse results with greater input quantity.
- Integrated training data yields better results with greater input quantity.
- Segregated training data yields worse results with greater look ahead distance.
- Integrated training data yields better results with greater look ahead distance.

Figure 4.8 and Figure 4.9 confirm these trends for the integrated training samples using 30 runs for statistical significance.

The problem we are facing is that the segregated training and the integrated training have complete opposite trends with respect to how number of inputs and look ahead distance affects the performance. These events can be understood using the same explanations as in the previous section.

Table 4.6: All raw results for “no limits” analysis

100 look ahead window

	Segregated Training				Integrated Training			
Configuration	4inputs	6inputs	8inputs	10inputs	4inputs	6inputs	8inputs	10inputs
no learning	0.147				0.127			
elev	−169%	−172%	−220%	−172%	**30.1%**	**28.5%**	**34.9%**	**35.8%**
grade	9.2%	8.8%	5.5%	−10.5%	16.9%	15.9%	**21.5%**	**20.5%**
slopes	7.7%	9.4%	−2.6%	−4.7%	12.5%	10.6%	12.0%	10.5%
bridge	7.7%	7.2%	0.3%	3.3%	11.6%	11.5%	14.5%	12.7%
vdes	−6.6%	−4.6%	−13.3%	−17.3%	4.3%	1.3%	8.1%	10.0%
vdesslopes	−0.7%	−0.2%	−3.7%	−192.9%	1.3%	1.5%	1.3%	1.8%
vdesbridge	−1.2%	0.7%	−2.5%	−4.3%	1.7%	1.4%	1.9%	1.3%

400 look ahead window

	Segregated Training				Integrated Training			
Configuration	4inputs	6inputs	8inputs	10inputs	4inputs	6inputs	8inputs	10inputs
no learning	0.144				0.124			
elev	−117%	−141%	−295%	−173%	**39.6%**	**38.8%**	**49.2%**	**50.0%**
grade	−7.2%	−3.6%	−13.9%	−16.2%	**28.1%**	**25.7%**	**39.6%**	**36.6%**
slopes	−15.8%	−25.1%	−35.5%	−34.2%	19.1%	**25.2%**	**29.1%**	**34.9%**
bridge	−19.2%	−9.8%	−33.6%	−31.3%	**23.2%**	**26.2%**	**33.2%**	**35.5%**
vdes	−8.6%	−20.3%	−29.9%	−26.0%	13.4%	16.2%	**30.3%**	**31.7%**
vdesslopes	−12.3%	−18.5%	−25.2%	−23.4%	3.4%	1.6%	6.2%	7.0%
vdesbridge	−12.2%	−16.0%	−19.3%	−26.6%	3.4%	2.6%	7.0%	5.7%

800 look ahead window

	Segregated Training				Integrated Training			
Configuration	4inputs	6inputs	8inputs	10inputs	4inputs	6inputs	8inputs	10inputs
no learning	0.142				0.122			
elev	−586%	−586%	−213%	−601%	**45.0%**	**45.8%**	**60.6%**	**62.8%**
grade	−12.2%	−18.1%	−35.4%	−40.3%	**25.2%**	**32.0%**	**49.4%**	**49.9%**
slopes	−59.1%	−32.2%	−34.4%	−104%	**30.1%**	**30.0%**	**47.0%**	**48.7%**
bridge	−45.2%	−55.6%	−75.0%	−70.4%	**30.0%**	**32.4%**	−572%	**47.3%**
vdes	−24.8%	−21.9%	−64.2%	−64.0%	**25.6%**	**36.7%**	**52.2%**	**54.5%**
vdesslopes	−22.1%	−25.2%	−48.6%	−51.6%	6.9%	6.1%	**25.2%**	**27.7%**
vdesbridge	−17.5%	−20.6%	−44.2%	−45.0%	3.5%	4.5%	18.7%	**26.2%**

Segregated training only gives a subsect of the possible terrains to train on. If that data is learned too well, then it will not test well on new data. However, if it doesn’t learn enough, it will also perform poorly. We can tell which extreme is

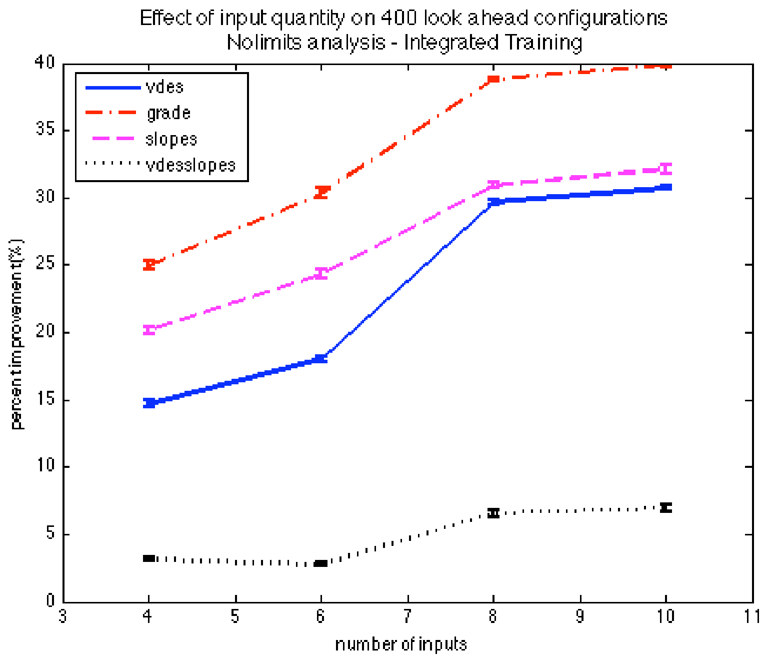

Figure 4.8: Impact of input quantity on percent improvement in “no limits” analysis : Different configurations, all with a look ahead distance of 400 meters, were ran through the neural network 30 times with 4, 6, 8, and 10 inputs.

occurring by the trends. By adding more inputs, we are inputting more information about the elevation or desired velocity curve to the neural network. It is possible that greater amounts of inputs defines too specific of a curve and fewer inputs generalize the terrain better, therefore making it easier to implement on new data. However, in the integrated training examples. More specific is better because it will be tested on data that is very similar to the training data. The same argument can be made for look ahead distance. The explanation for the very poor result of the ”bridge” configuration, 8 input data point, is that the system most likely hit a

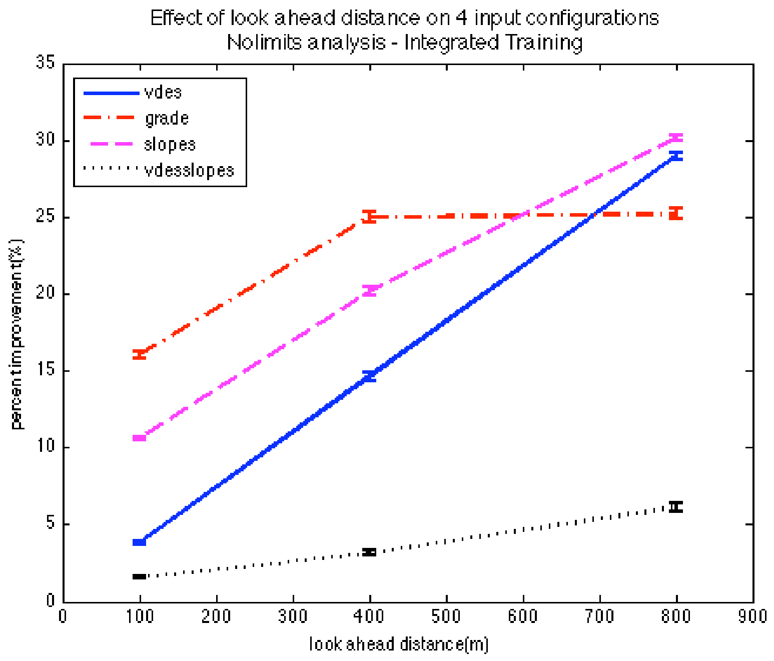

Figure 4.9: Impact of look ahead distance on percent improvement in “no limits” analysis: Different configurations, all with 4 inputs, were ran through the neural network 30 times with 100, 400, and 800 meter look ahead distances.

local minima in its training, resulting in bad neural network weights and therefore a bad solution.

Even if integrated training is not best for robustness, these “no limits” analysis results are excellent. An improvement of up to 60% would benefit this system greatly.

Chapter 5 – Multi-Concept Inputs

Although the single concept inputs have produced very beneficial results, in many cases a multi concept approach can improve the performance of a learner.

Two multi-concept inputs were explored:

Enriched Input Space: This method enriches the input space by including inputs from both the elevation based configurations and the desired velocity based configurations.

Ensemble: This method combines the results from several of the single-concept neural networks. In this method, not just one input from each input space will be used, but up to 28 different input representations will be incorporated into one solution.

First we will briefly discuss the results from the enriched input space method. Lastly, we will examine the benefits and disadvantages of ensemble solutions.

5.1 Enrich Input Space

By enriching the input space we are simply adding more types of inputs to the neural network. For example, in our single concept 4 input configurations, 1 piece of information is given at 4 points in the future. Now in our new enriched input space, we will input 2 pieces of information each at 4 points in the future for a total of 8 inputs. For this study, we have two types of information easily on hand in the form of the GPS acquired elevation curve and the PCC acquired desired velocity curve. Using the 4 input configurations from before, we will input the two types of information into the neural network together. Figure 5.1 demonstrates this idea.

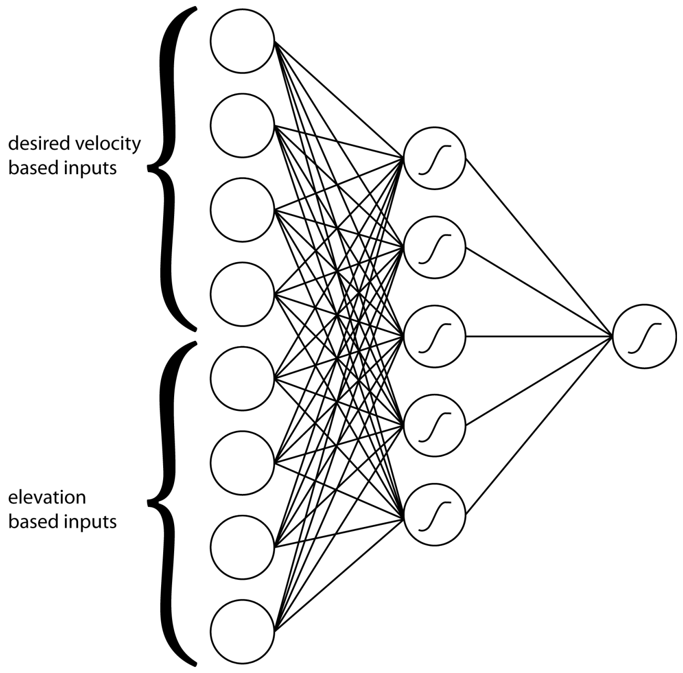

Figure 5.1: Multi-Concept learning: Learning can occur with two different types of inputs. It is possible that with information from both the desired velocity curve and elevation curve, better learning can be realized.

Intuition makes us believe that if a neural network can learn on one type of information, then more information will result in greater learning ability. This makes since to us as humans because our own brains are able to link together multiple pieces of knowledge to quickly make conclusions. This is not always true for a neural network. Sometimes, when different types of information are entered together into the same neural network, the network can interpret the extra information as noise [16]. This kind of result is not a certainty, so it is a good method to experiment with as better learning can occur. However, this concern will help to choose which inputs to use.

Three different input pairings were chosen for testing. Each of these pair a elevation based configuration input with a desired velocity based configuration input. Following are the input pairings and the reason behind its selection:

1. **“vdes & grade”:** This pairing was chosen based on their superior performance in the single concept results.

2. **“vdesslopes & slopes”:** We want to try to avoid confusing the neural network. Although these inputs come from different data, they are in the same form, which may help the neural network make the correct connections for learning.

3. **“vdesbridge & bridge”:** This pairing was chosen as a variation of a “same form” input type

Results for each run are presented in Table 5.1. Networks were all run with 40 hidden units. Four inputs of each type were used for a total of 8 inputs to the system. Columns are divided into 100, 400 and 800 look ahead window data. For comparison, the best single concept improvement from a single run is tabulated

Table 5.1: Enriched input space results: Raw results with various multi-concept configurations

Basic analysis

	Segregated Training			Integrated Training		
Configuration	100 look	400 look	800 look	100 look	400 look	800 look
no learning	0.225	0.225	0.225	0.195	0.195	0.195
"vdes & grade"	**25.8%**	11.53%	**5.1%**	0.6%	−0.9%	−3.8%
"vdesslopes & slopes"	**25.4%**	0.5%	−6.0%	5.9%	0.4%	−1.6%
"vdesbridge & bridge"	25.0%	−9.4%	−3.8%	3.0%	−0.3%	−3.5%
best single concept	25.0%	26.7%	−0.2%	43.9%	32.7%	11.8%

"No limits" analysis

	Segregated Training			Integrated Training		
Configuration	100 look	400 look	800 look	100 look	400 look	800 look
no learning	0.147	0.144	0.142	0.127	0.124	0.121
"vdes & grade"	−10.6%	−32.3%	−44.9%	**50.1%**	**59.6%**	62.8%
"vdesslopes & slopes"	−0.7%	−24.7%	−69.1%	**32.7%**	**51.2%**	58.8%
"vdesbridge & bridge"	−0.4%	−14.4%	−78.1%	27.9%	44.6%	57.8%
best single concept	9.4%	−3.6%	−18.1%	35.8%	49.2%	62.8%

under the multi-concept configurations. Values are in bold if they exceed the improvement of the best single concept result.

In the basic analysis the multi concept solutions exceeded the best single concept solution only 3 times and each of these times were by inconsequential amounts. The "no limits" analysis has some results of interest. With integrated training and 100 or 400 meter look ahead windows, the "vdes & grade" and the "vdesslopes & slopes" configurations have exceeded the best single concept result. The former configuration does better than the latter in these instances. This result is worth following, however multi-concept systems are more complex and if it were to be utilized, a cost analysis would need to be conducted to see if it really does perform better than the single concept networks.

5.2 Ensemble Methods

The concept of ensemble started for the purpose of classification. The idea is that many systems using different inputs will output a class prediction and then vote upon what classification to select. This vote can perform better than a single system. Applied to this study, the ensemble will average the solutions of each system so as to let each system contribute to the ensemble solution. The only way this will be successful is if each of the systems have made different errors [2, 3, 17, 25, 28, 33, 40]. For example, if the elevation based inputs tend to overshoot the target value during the downhill terrains, then we hope that at the same position the desired velocity based inputs undershoot so that the average is closer to the target than any one of the single concept inputs. Figure 5.2 demonstrates this concept.

Selecting how to combine the systems into different sized ensembles is not too difficult. Ideally, all the systems would be used; however, knowing some of the results from the single concept section, ensemble inputs can be selected intelligently. The total number of systems that can be used for the ensemble is 28, given 7 configurations and 4 choices of input quantity. Runs of different training method, and look ahead distance cannot be ensembled because they do not share common testing samples. From the single concept results, the "elev" configuration performs terribly with segregated training. Therefore, each of these 4 systems are not used during the segregated training runs only. Another observations from the single concept results is that configurations "vdesslopes" and "vdesbridge" often

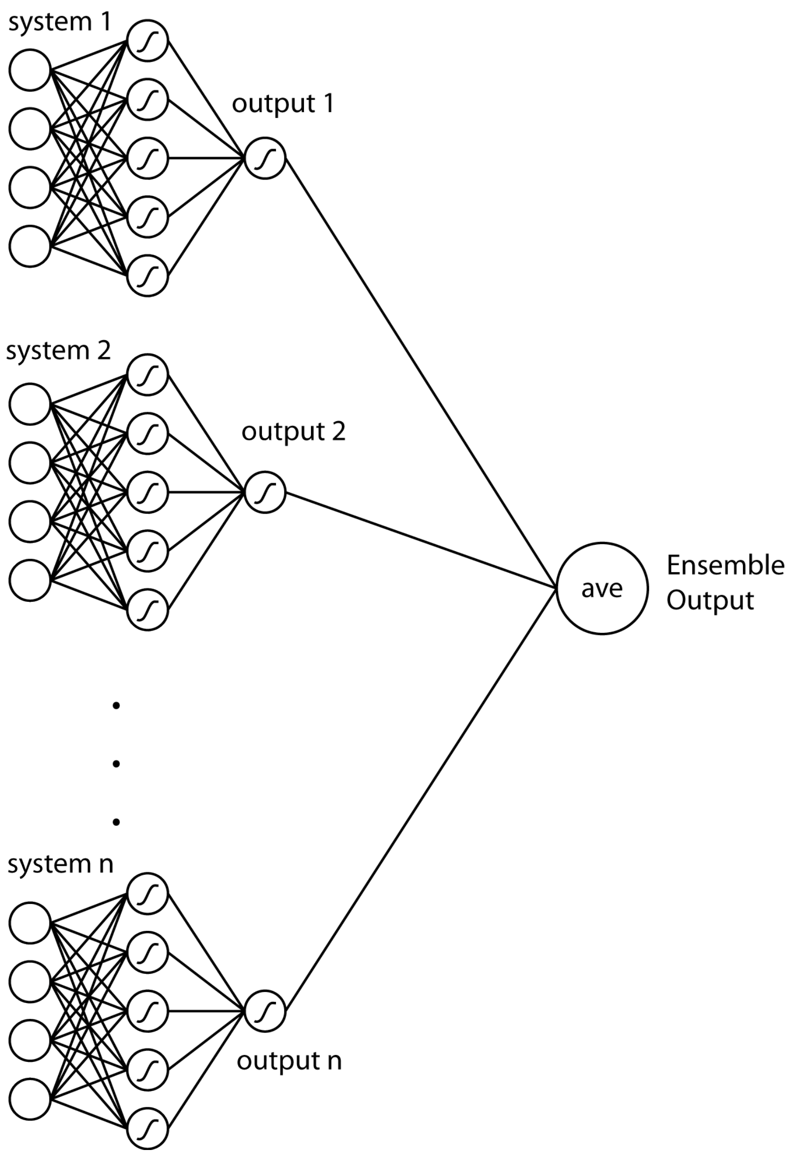

Figure 5.2: Ensemble Learning: The outputs of several systems can be averaged in the hopes that this cooperative solution is closer to the target than any one of the single-concept systems.

perform below average. Therefore, for one of the tests in each training method, these configurations were omitted. Consider that each configuration contributes 4 systems corresponding to that configuration analyzed with 4, 6, 8,and 10 inputs.

The ensembles tested are as follows:

- For 100, 400 and 800 look ahead distances
    - Segregated Training
        - A 16 system ensemble omitting “elev”,“vdesslopes”, and “vdesbridge” configurations
        - A 24 system ensemble omitting “elev” configurations
    - Integrated Training
        - A 20 system ensemble omitting “vdesslopes”, and “vdesbridge” configurations
        - A 28 system ensemble with all configurations at all input quantities

Table 5.2 shows the results for these tests using the basic analysis and “no limits” analysis. The first column of each training method is labeled “no learning” and is the average meters/second offset between the actual velocity and PCC’s desired velocity. All other cells represent the percent improvement over the no learning offset from a single run. The different learners shown are those stated above, as well as the best run from the single concept networks. These values can be extracted from Table 4.5 for the basic analysis or Table 4.6 for “no limits” analysis.

Table 5.2: Ensemble results: Raw results with various ensemble sizes

Basic analysis

	Segregated Training				Integrated Training			
Ensemble Size	no learning	16 systems	24 systems	best single	no learning	20 systems	28 systems	best single
100 look ahead	0.225	21.3%	21.4%	25.0%	0.195	37.3%	28.9%	43.9%
400 look ahead	0.225	**30.8%**	**30.6%**	26.7%	0.195	28.0%	25.9%	32.7%
800 look ahead	0.225	**25.7%**	**20.5%**	−0.2%	0.195	−0.7%	4.2%	11.8%

"No limits" analysis

	Segregated Training				Integrated Training			
Ensemble Size	no learning	16 systems	24 systems	best single	no learning	20 systems	28 systems	best single
100 look ahead	0.147	**11.5%**	**10.8%**	9.4%	0.127	24.9%	21.3%	35.8%
400 look ahead	0.144	**-2.1%**	**-1.9%**	−3.6%	0.124	43.5%	38.6%	50.0%
800 look ahead	0.142	**-10.9%**	**-7.6%**	−18.1%	0.121	42.9%	45.7%	62.8%

In the basic analysis we see that ensemble methods using segregated training performs the best at 400 meters look ahead distance and about the same at 100 and 800 meters look ahead distances. We also see that the 16 system ensemble and the 24 system ensemble have very similar improvement values. A very interesting result occurs in the 400 and 800 meter look ahead rows. The ensembles from both segregated and integrated training methods perform better than any of the single concept networks within their own look ahead window. In fact, the 400 look ahead ensemble performs better than any other non-smoothed run using segregated training.

In the "no limits" analysis we notice the same trends as with the single concepts. That is, the segregated training method has better results with shorter look ahead distances and the integrated training method has better results with longer look ahead distances. Omitting the weaker networks from the ensemble sometimes helped and sometimes hurt the system. However, the best result seen includes all the networks. This result is found with the 800 look ahead, integrated learning run. We also see that ensemble reaches higher improvement percentages than the single concepts using segregated training but not integrated training.

Recall that in Table 4.6 the 800 look ahead, 8 input, “bridge” configuration in “no limits” analysis had a poor result due to a faulty training session. Removing this network from the ensemble results in a 27 system ensemble with an improvement of 51.4%. This result was plotted next to the best “no limits” result from the single concept network: The “elev” configuration with 10 inputs and integrated training. This solution yielded a single run improvement of 62.8%. See Figure 5.3 for the comparison.

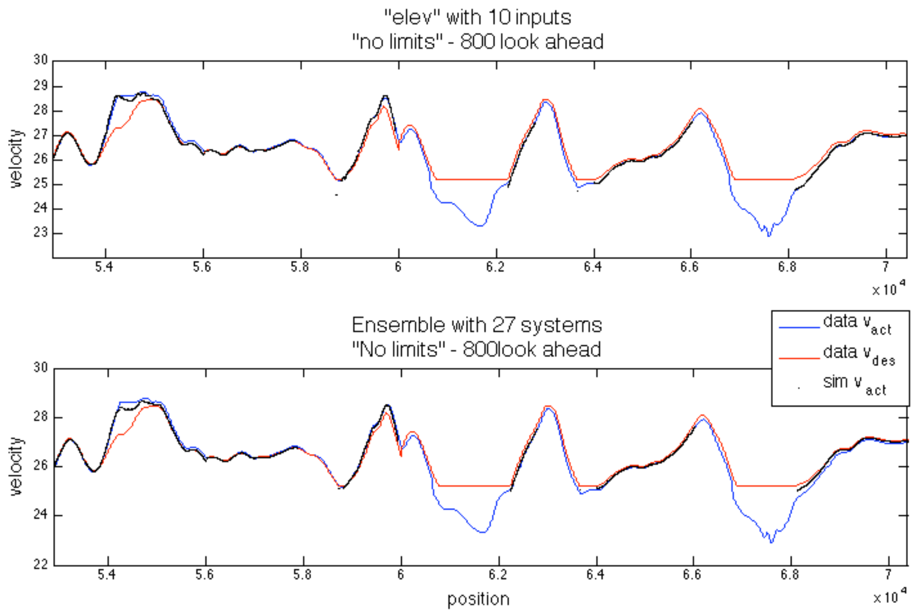

Figure 5.3: **top:** The best solution found for a single concept neural network. **bottom:** The best ensemble solution found for “no limits” analysis. A comparison between the best solutions from single concept and ensemble methods.

These plots show that they both look like very good solutions compared to the others we have seen. One thing to notice is that with either of these, smoothing would do little benefit. In reviewing the ensemble results it is important to note that averaging networks is a form of smoothing the solution. Therefore, while many of the single concepts raw results have room for improvement in the form of smoothing, the ensembles does not have this opportunity as it is already smoothed.

A hinderance for ensemble is that it requires 16 or more networks to be run rather than just one. Therefore, the pay off has to counter balance the additional computational cost.

Chapter 6 – Conclusion

The work presented in this thesis aims to find an input representation that improves the performance of a pre-existing model based controller using predictive cruise control technologies. The biggest challenge in this thesis was to find the input mapping that best predicts the PCC desired velocity offset from the actual vehicle velocity. Hundreds of different input variations were analyzed. These variations varied input parameters such as input quantity, look ahead distance, and configuration, as well as training and testing methods such as segregated training, integrated training, and “no limits” analysis.

Although there were many results that did not perform well, there were a few input representations that consistently produced promising results and deserve further attention.

6.1 Contributions of this Thesis

In applying neural network learning in combination with the PCC model based learner, we have found that the learner can benefit the system by up to about 60% improvement. This success is contingent upon several trends that were seen from experimentation with different networks. Perhaps the most interesting results come from the differences between segregated and integrated type training methods.

Every trend seems to be opposite for integrated training than it is for segregated training. For example, the affect of input quantity and look ahead distance. This makes it difficult to tell what parameters are good and which are bad. Ultimately, the performance of the parameter is effected by the training method. Integrated training, as expected, is able to get better results because the training data is very similar to the testing data. The segregated training method will identify the more robust input mappings. Overall, the training data that is available and the terrain that the vehicle will be subject to, will determine what network will be chosen.

Furthermore, this study effectively discovered what does **not** work. Several of the configurations including “elev”, “vdesslopes”, “vdesbridge”, and to a lesser extent, “slopes”, do not perform up to an adequate performance. We also know that to get decent results for the “no limits” analysis, we must have a very complete training set that represents the terrain the vehicle is expected to travel. Given the segregated training, the “No limits” solutions performed awfully, while the integrated training solutions exceeded expectations.

Multi-Concept solutions were most often not worth the added complexity and computational time. However, there were a few instances that easily bested even the highest single concept performance within the same learning method and look ahead distance categories.

The conclusions drawn from this thesis include:

1. Success rates are highly dependant on training method. To obtain the best results possible, availability of relevant data sets will be vital to the success of the learner if implemented into a commercial vehicle.

2. Segregated training and Integrated training represent the two extremes of training: not very representative and very representative of full data set, respectively. Therefore, a more moderate training method may be beneficial.

3. Elevation based inputs typically produce better results than the desired velocity based inputs. Within the elevation based inputs, the “grade” configuration was the most consistent performer. This means that the grade of the road is one of the more relevant attributes in finding the PCC offset.

4. Multi-concept methods of learning have potential but a cost analysis must be conducted to know its true benefit.

From these conclusions we derive 3 significant contributions of this thesis. All of these contributions stem from improving the Predictive Cruise Control and result in saving both fuel and money.

The main contributions that this thesis has shown are as follows:

1. **Improvement upon PCC by implementing learning control:** Improvement in the total system of up to 60% can be made by implementing a learner into the Predictive Cruise Control algorithm. This improvement will ultimately reduce errors in PCC, resulting in: 1) An increase in the controllers ability to reduce fuel consumption and 2) A potential increase in the recalculation distance, thus reducing processing power required.

2. **Extraction of relevant input configurations:** This thesis' results reveal which environmental attributes contribute most to the PCC model offset.

For example, “grade” and the other elevation based input mappings performed the best in the single concept results. For the multi concept results, “grade” paired with “vdes” configurations performed the best. Knowing which configurations perform well will lead to the development of a more effective predictive cruise controller.

3. **Indication of training method and data set suited for practical implementation:** In analyzing results with different training methods, we have discovered the important role that the training set will play during the practical implementation of this controller. Given what data is available to train on, we can predict how much the learner can benefit the PCC system on a given road. For example, if the training data is very representative of the road the truck will be driving on, then we can predict that the learner will benefit the system by up to 60%.

Having demonstrated that learning methods are a good fit for this problem, there are several options open for future work in this area.

6.2 Future Work

The future work for this study in collaboration with Daimler Trucks North America, will be geared toward implementation of a learner into PCC for vehicle control. Improving the “no limits” analysis will be a major focus. As of now, there are short term goals and long term goals:

Short Term Goals: There are several short to mid ranged goals that will need to be completed before this technology makes it into commercial vehicles.

1. Now that a extensive search for valid inputs has been conducted it would be beneficial to narrow down the input representation variants to about 4 or 5 of the most promising mappings.
2. Once narrowed down, much more in-depth experiments can be ran on these network parameters and input configurations. Based on the results from this study, the "grade" and "vdes" configurations would be obvious choice for configuration.
3. Integrated and Segregated training methods are two training extremes. Several more moderate training approaches like segmented training method, can be analyzed to see more realistic results.
4. Analyze the pay-off/cost of Multi-concept learning methods to determine if this thread should be pursued.
5. Test other options such as hierarchial decision making to choose a control method based on the upcoming terrain. Also, other learners such as reinforcement learning could be explored.
6. Optimize the performance and robustness of the network.

Long Term Goals: The long term goals of this project are to:

1. Implement the learner into the PCC controller

2. Apply the network into a real world situation(i.e. in a truck's cruise control system)

Control of a commercial vehicle for fuel optimality is a very interesting and current topic. This thesis has shown that a learning approach can benefit the robustness of vehicle control and that there is great potential for the future of this technology.

Bibliography

[1] J. N. Amaral, J. Ghosh, and K. Tumer. Applying genetic algorithms to the state assignment problem: A case study. In *SPIE Conf. on Adaptive and Learning Systems, SPIE Proc. Vol. 1706*, pages 2–13, Orlando, Fl., April 1992.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] H.-J. Chang, J. Ghosh, and K. Liano. A macroscopic model of neural ensembles: Learning induced oscillations in a cell assembly. *International Journal of Neural Systems*, 3(2):179–198, 1992.

[4] F.-C. Chen. Back-propagation neural network for nonlinear self-tuning adaptive control. *Intelligent Control, 1989. Proceedings., IEEE International Symposium*, pages 274–279, Sep 1989.

[5] L. C. Davis. Effect of adaptive cruise control systems on traffic flow. *Phys. Rev. E*, 69(6):066110, Jun 2004.

[6] L. C. Davis. Effect of adaptive cruise control systems on mixed traffic flow near an on-ramp, 2005.

[7] L. C. Davis. Driver choice compared to controlled diversion for a freeway double on-ramp, 2008.

[8] M. Dorigo and L. M. Gambardella. Ant colony systems: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[9] H. Drucker, C. Cortes, L. D. Jackel, Y. LeCun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.

[10] E. Frazzoli, M.A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. volume 1, pages 821–826, Sydney, NSW, Australia, Dec 2000. Decision and Control, 2000.Proceedings of the 39th IEEE Conference.

[11] P. Gaudiano, E. Zalama, and J.L. Coronado. An unsupervised neural network for low-level control of a wheeled mobile robot: noise resistance, stability, and hardware implementation. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions*, 26(3):485–496, Jun 1996.

[12] Douglas C. Giancoli. *Physics for Scientists and Engineers*. Pearson, 3^{rd} edition, 1999.

[13] S. Hashem. Effects of collinearity on combining neural ensembles. *Connection Science, Special Issue on Combining Artificial Neural Networks: Ensemble Approaches*, 8(3 & 4):315–336, 1996.

[14] Erik Hellström, Maria Ivarsson, Jan Åslund, and Lars Nielsen. Look-ahead control for heavy trucks to minimize trip time and fuel consumption. Fifth IFAC Symposium on Advances in Automotive Control, Monterey, CA, USA, 2007.

[15] Ronaldo R. Torres Jr., Jose' L. Silvino, Paulo F. M. Falmeira, and Julio C. D. de Melo. Control of autonomous robots using genetic algorithms and neural networks. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 151–157, October 1999.

[16] J. Junell, M. Knudson, and K. Tumer. Optimizing sensor / neuro-controller pairings for effective navigation. *Artificial Neural Networks in Engineering*, 2008.

[17] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation and active learning. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems-7*, pages 231–238. M.I.T. Press, 1995.

[18] Kristof Van Laerhoven, Kofi A. Aidoo, and Steven Lowette. Real-time analysis of data from many sensors with neural networks. In *IEEE Fifth International Symposium on Wearable Computers*, page 115, October 2001.

[19] Julien Laumônier, Charles Desjardins, and Brahim Chaib-draa. Cooperative adaptive cruise control: A reinforcement learning approach. In *4th Workshop on Agents in Traffic And Transportation, AAMAS'06*, Hakodate, Hokkaido, Japan, May 2006.

[20] Tsang-Wei Lin, Sheue-Ling Hwang, Jau-Ming Su, and Wan-Hui Chen. The effects of in-vehicle tasks and time-gap selection while reclaiming control from adaptive cruise control(acc) with bus simulator. In *Accident Analysis and Prevention*, pages 1164–1170, May 2008.

[21] W. G. Macready and D. H. Wolpert. Bandit problems and the exploration/exploitation tradeoff. *IEEE Transactions on Evolutionary Computation*, 2:2–22, 1998.

[22] Greg Marsden, Mike McDonald, and Mark Brackstone. Toward and understanding of adaptive cruise control. *Transportation Research Part C: Emerging Technologies*, 9(1):33–51, 2001.

[23] Konstantin Neiss, Thomas Connolly, and Stephan Terwen. Patent 6990401: Predictive speed control for a motor vehicle, January 2006.

[24] Hiroshi Ohno, Toshihiko Suzuki, Keiji Aoki, Arata Takahasi, and Gunji Sugimoto. Neural network control for automatic braking control system. *Neural Netw.*, 7(8):1303–1312, 1994.

[25] D. W. Opitz and J. W. Shavlik. Actively searching for an effective neural network ensemble. *Connection Science, Special Issue on Combining Artificial Neural Networks: Ensemble Approaches*, 8(3 & 4):337–354, 1996.

[26] D. W. Opitz and J. W. Shavlik. Generating accurate and diverse members of a neural-network ensemble. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems-8*, pages 535–541. M.I.T. Press, 1996.

[27] Kevin M. Passino. *Biomimicry for Optimization, Control, and Automation.* Springer-Verlag, London, 2005.

[28] M.P. Perrone and L. N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone, editor, *Neural Networks for Speech and Image Processing*, chapter 10. Chapmann-Hall, 1993.

[29] R. Rajamani and C. Zhu. Semi-autonomous adaptive cruise control systems. *IEEE Transactions on Vehicular Technology*, 51(5):1186–1192, 2002.

[30] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, Inc., Upper Saddle River, New Jersey 07458, 2003,1995.

[31] Roland Schulz and Kay Fürstenberg. *Advanced Microsystems for Automotive Applications 2006*. Springer Berlin Heidelbergh, 2006.

[32] Bobbie D. Seppelt and John D. Lee. Making adaptive cruise control (acc) limits visible. *International Journal of Human-Computer Studies*, 65(3):192–205, 2007.

[33] A. J. J. Sharkey. (editor). *Connection Science: Special Issue on Combining Artificial Neural Networks: Ensemble Approaches*, 8(3 & 4), 1996.

[34] P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems-8*, pages 190–196. M.I.T. Press, 1996.

[35] N.A. Stanton, M. Young, and B. McCaulder. Drive-by-wire: The case of driver workload and reclaiming control with adaptive cruise control. *Safety Science*, 27(2):149–159, 1997.

[36] R.S. Sutton and A.G. Barto. *Reinforcement Learing: An Introduction*. MIT Press, 1998.

[37] K. Tumer and A. Agogino. Robust coordination of a large set of simple rovers. In *Proceedings of the IEEE Aerospace Conference*, Big Sky, MO, March 2006.

[38] K. Tumer, K. D. Bollacker, and J. Ghosh. A mutual information based ensemble method to estimate the bayes error. In C. *et al.* Dagli, editor, *Intelligent Engineering Systems through Artificial Neural Networks*, volume 8, pages 17–22. ASME Press, 1998.

[39] K. Tumer and J. Ghosh. Limits to performance gains in combined neural classifiers. In *Intelligent Engineering Systems through Artificial Neural Networks*, volume 7, pages 419–424, St. Louis, 1995.

[40] K. Tumer and J. Ghosh. Linear and order statistics combiners for pattern classification. In A. J. C. Sharkey, editor, *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, pages 127–162. Springer-Verlag, London, 1999.

[41] K. Tumer and J. Ghosh. Robust order statistics based ensembles for distributed data mining. In H. Kargupta and P. Chan, editors, *Advances in*

Distributed and Parallel Knowledge Discovery, pages 185–210. AAAI/MIT Press, 2000.

[42] A. Vahidi and A. Eskandarian. Research advances in intelligent collision avoidance and adaptive cruise control. *IEEE Transactions on Intelligent Transportation Systems*, 4(3):143–153, 2003.

[43] D. H. Wolpert and K. Tumer. Optimal reward functions in distributed reinforcement learning. In *Proceedings of the Second Asia-Pacific Conference on Intelligent Agent Technology (IAT-2001)*, Maebashi City, Japan, 2001.

[44] D. H. Wolpert, K. Wheeler, and K. Tumer. General principles of learning-based multi-agent systems. In *Proceedings of the Third International Conference of Autonomous Agents*, pages 77–83, 1999.

[45] D. H. Wolpert, K. Wheeler, and K. Tumer. Collective intelligence for control of distributed dynamical systems. *Europhysics Letters*, 49(6), March 2000.

[46] An-Min Zou, Zeng-Guang Hou, Si-Yao Fu, and Min Tan. Neural networks for mobile robot navigation. a survey. In Jun Wang, Zhang Yi, Jacek M. Zurada, Bao-Liang Lu, and Hujun Yin, editors, *Advances in Neural Networks - ISNN 2006*. Springer, 2006.