Introduction

The NHL is an incredibly interesting professional sports league because of the parity between all teams in the league. It is not often that the best regular season team in the league is able to win the Stanley Cup, which makes it incredibly difficult to predict who will win the championship for both humans and computers alike. Many previously published predictive models for the NHL playoffs only attain approximately 60% accuracy. For this project, three different neural networks have been put to the task of predicting the total post season win percentages of teams that make it to the Stanley Cup Playoffs. While we are interested in optimizing the models individually, what we are more concerned with is comparing 3 network models applied to the same task.

Problem Formulation

The data for this problem has been sourced directly from the NHL.com website. The input data is the aggregate data for each team over an entire season, including some advanced statistics like face off win percentage, penalty kill/power play percentage and more. Because of changes to NHL rules in the last 30 years, only the previous 21 seasons have been used to create the data set for this model. The ground truth labels are the total post season win percentage per team. That is:

$$total\ post\ season\ win\ percent = \frac{wins\ by\ team\ X}{total\ games\ played\ in\ post\ season\ by\ all\ teams}$$

These labels were chosen because they can be used to train a soft-max probability output for a Stanley Cup Winner prediction task. A probability output like soft-max is more beneficial to real world applications like sports betting, rather than treating this as a classification task to select a singular championship winner.

The Stanley Cup Playoffs always include 16 teams. We have pulled the last 21 years of NHL regular season and post season data; therefore, this leaves us with 336 data samples.

Approaches and Baselines

For this task we define the baseline prediction to be a naïve approach in which the post season winning percentage of each team is based directly on the finishing place of the team in the regular season. The bottom 8 finishing teams are predicted to win $1/64^{th}$ of all games in the post season, teams finishing in 8^{th} to 4^{th} place are predicted to win 1/16, teams finishing in 4^{th} to 2^{nd} place are predicted to win $1/8^{th}$, and the two top teams in the regular season are predicted to win $3/16^{th}$ of all games in the post season.

We have applied 3 different neural nets to attempt to solve this task: a feed forward network for analysing teams independently, a feed forward network for analysing teams as a cohort per year, and a 1-dimensional convolutional neural network which analyses teams as a cohort as well. All neural networks have been implemented using TensorFlow. For these models, the hyper

parameters are the number of neurons in the hidden layers, the activation function for each neuron, and the optimization function used to train the neural network.

The structure of the neural net for analyzing teams independently is an input layer of 15 neurons, a hidden layer, and a single linear output neuron.

The batched feed forward network has the following structure: first, flatten the data to make an input layer of 240 neurons (16 teams * 15 features per team). This input feeds into a fully connected hidden layer, and finally our output layer is a 16-neuron soft-max layer.

The 1-dimensional convolutional neural network uses the convolutional layer as the input layer, which feeds directly into a global averaging layer. Again, a 16-neuron soft-max output layer is used.

The feed forward neural network was trained by breaking the data into 280 entries for training, and 56 entries for testing. The training data is then further broken down for hyper parameter tuning using a k-folds validation algorithm.

The other two neural networks have a much lower number of data sets since16 teams are batched together at a time. Because there are only 21 possible inputs and labels, we have used one set of inputs and labels as a test set, and the other 20 inputs and labels are used to train the model. Hyper parameters are tuned using a hold out one algorithm because of the small number of cohorts.

Evaluation Metric

Given that this is a regression-like task, the measure of success is given as mean absolute error and the training objective is mean square error. Due to the minimal available data, convexity of the training objective was an important feature.

Results and Discussion

All estimates are given as decimals on [0, 1] and are intended to represent an estimate of a team's total winning percentage in the post season. Here, results are given as the percentage, rather than the decimal representation. The naïve guess gives us a mean absolute error of 5.08%. Given that for any post season, the winning team usually would have a total post season winning percentage of 18%-22%, this is a pretty abysmal guess.

Analyzing individual teams using a feed forward neural network yields only a slightly better result. The mean absolute error for this model is approximately 4.3%. The best structure for this simple feed forward network is a 15-neuron hidden layer with ReLU activation functions and optimized using stochastic gradient descent.

Batching teams into the years in which they competed logically makes a lot more sense for this task. Batching is a better approach because we are assuming there are indicators in the advanced

data which might indicate a team is capable of beating any other team. For example, perhaps power play and penalty kill percentages are more important to winning the Stanley Cup than total regular season points.

The batched soft-max neural net performed best with a hidden layer of 128 neurons and using a linear activation function. For this network, an Adam stochastic gradient descent algorithm provided the best optimization. The mean absolute error for this model is typically approximately 2.6%. This was the best performance for any of the three simple networks.

Using this same batching technique but applying a simple 1-dimensional convolutional network seemed like logical solution to this problem, however, the performance of the convolutional model was not exceptional. Given optimal hyperparameters: a kernel size of 8, ReLU activation functions and stochastic gradient descent in order to optimize the network - the model was only able to attain a mean absolute error of approximately 4.1%. Analysis of the training of the model shows that it has an issue with overfitting, however, introducing a dropout layer did not improve the performance at all.

In order to improve any of these models, more data, both advanced statistics, and the number of seasons used to train the models, would be very helpful.