NHL PLAY-BY-PLAY PREDICTION

PROJECT REPORT

Timur Garipov
tgaripov@mit.edu

Adam Kaplan
akapl@mit.edu

Alexander Lynch
ajlynch@mit.edu

December 10, 2019

1 Introduction

There are a lot of sport papers floating around, especially concerning the NBA and MLB. While we focus on the not so explored NHL, we come at it with a different motivation. Our goal for this paper is threefold. First, we predict the outcome of an NHL game for a given team. Second, we predict the score difference (Home - Away) of a game. Third, we predict these outputs over the course of a game rather than once before the game starts.
Our motivation for this is to come up with a way to analyze a game as it is happening instead of just predicting the outcome ahead of time. To do that, we extend research on using mixture density networks in predicting score differences in NBA games [1] to recurrent neural networks. We then evaluate the impact for each of the in-game events such as goals, hits, shots, on the win probability and use the mixture density to analyze the distribution over the score difference and what it tells us about the win/draw/lose probability throughout a game. We conclude by implementing a basic data augmentation methodology [2] in hopes of improving overall accuracy, but instead found that it does not necessarily improve it, but changes the model flexibility over time.

2 Data

We chose to use the NHL game data set [3]. This data set has a number of different files each of which contains statistics based on different levels of the game(game level, play level, player level, etc) as can be seen in figure 1. In total the data set contains 37 589 periods from 11 434 games. With 33 different teams this comes to around 1 000 period samples per team. We have two questions that we want to try to answer with this data set. The first is "What will be the outcome of the game for a given team?", and the second is "What will be the score differential of the game?" Additionally we will be asking these questions after every play of a game in order to add a slight change to the classic game prediction model as well as to answer a third question of "How does our model's predictions change over time?" In order to answer these questions we will need two different models which will be discussed in section 4. However both of these models will be able to make answer their respective questions from the same inputs.

Overall we want our input to be a combination of the plays in a time series format as well as a history of the teams recent game performance. In order to get a play by play input we use the `game_plays.csv` file which is composed of every play of every game from the past six seasons totaling over 3.6 million plays. In this context a "play" is an event that occurs within one of the categories shown in the left column of table 3.

In addition, there is information about the location of the event on the ice rink, but we chose not to include this in our model. Out of these categories we are only concerned with the type of event that occurs, and in our pre-processing of the data, we create a one hot vector indicating which of the event types is occurring. An example of a data object from the play by play data file as well as the column descriptors can be seen in table 2.

play_id	game_id	play_num	team_id_for	team_id_against	event	secondaryType	...	periodType	...
2011030221_7	2011030221	7	1	4	Shot	Wrist Shot	...	REGULAR	...

Table 1: A sample row from the already pre-processed input data. In addition to the event level input, we also include an aggregation of stats over the past 20 games.

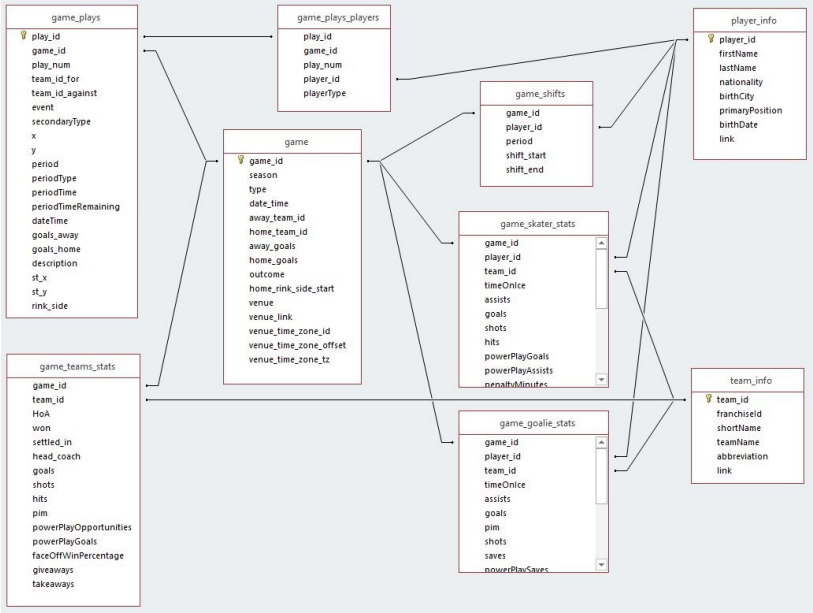

Figure 1: Original data layout [3].

The inclusion of game history is to account for relevant information of how the team is performing in hopes that the model can learn the relevance of past game performance to the current game. In order to put this information in a form for our model, we take data from the `game_team_stats.csv` which aggregates the stats for each team on a game by game basis totaling 37 589 games. Each entry in this file represents a single game for a single team.

TeamId	GameId	Won_History	Goals_History	Shots_History	Hits_History	PIM_History	PPOpportunities_History	PPGoals_History	FaceoffWP_History	Giveaways_History	Takeaways_History
1	2011030223	0.5	3.5	30.5	31.5	12	3.5	0.5	47.9	7	7

Table 2: A sample row from the game history data that has been aggregated over past games.

We combine the statistics for the games by averaging the entries along each column for the relevant games. Once we had created this aggregation, we ran it through a simple feed forward neural network to create an embedding of the aggregation which was then used as the final input alongside the one hot encoded event indicator. This embedding step was performed in order to reduce the dimensionality of the input. As we will discuss in 3.1 the aggregation embedding was also used when implementing data augmentation.

3 Approach

While we are interested in predicting the game outcome, our main interest is to see how each individual events (see table 3) in a game drive the prediction and how to measure the model performance over time. With that in mind we decided to use the mixture density approach used by Ganguly and Frank [1]. In their paper they train a Gaussian mixture density fully connected neural network on NBA score difference (Home - Away) instead of game outcome for three reasons. First, to take into account in-game events [1]. Second, to give a "measure of certainty [1]." Third, to make models more comparable [1]. Our implementation using a RNN was left as future work by the authors. The precise implementation is described in detail in section 4.

3.1 Data Augmentation

In order to expand the size of our data set we implemented a data augmentation scheme. The play by play data was a one-hot encoded vector, so the only option for data augmentation was the game history part of the input. Since the feature encoding of the game history is the actual inputs of the model, we chose to augment the encoding rather than the raw input. The augmentation technique we chose to use was adding zero centered Gaussian noise to each feature of the encoding as is described in [2]. The transformation can be written as

$$x_i' = x_i + \gamma \mathcal{N}(0, \varepsilon)$$

Game events	Past game statistics
Goal	Wins
Shot	Goals
Missed Shot	Shots
Blocked Shot	Hits
Hit	Penalty in minutes (PIM)
Face-off	Power play opportunities
Penalty	Power play goals
Takeaway	Face-off win percentage
Giveaway	Giveaways.
Official Challenge	Takeaways.

Table 3: List of game event types and past game statistics.

where x_i is the feature from the original data point, x_i' is the corresponding feature of the augmented data point, γ is a scaling factor, and ε is the standard deviation of the noise that is added. We found that the optimal values of γ and ε were 0.4 and 0.1 respectively.

3.2 Win/Draw/Lose probability

Finally, throughout this paper we will be referring to win/draw/lose probability and we want to be clear what is meant by that. Since the output of our model is a Gaussian mixture model on the score difference, it allows for an easy calculation of said probabilities. All of the probabilities are written in terms of the home team, i.e. "win" means home team wins. A positive score difference $S > 0$ means that the home team scored more goals than the away team, i.e. home won. Similarly for a negative score difference $S < 0$ home lost. Since score difference is continuous, the event $S = 0$, draw, would occur with probability 0. To account for that we decided to create a buffer around 0 of size 0.5. That means the event $S \in [-0.5, 0.5]$ represents a draw. The choice of 0.5 is arbitrary as long as it is greater than 0 and less than 1. Adjusting the win and lose events accordingly gives us,

$$\begin{aligned} \mathbb{P}(\text{Home win}) &= \mathbb{P}(S > 0.5), \\ \mathbb{P}(\text{Draw}) &= \mathbb{P}(S \in [-0.5, 0.5]), \\ \mathbb{P}(\text{Home lose}) &= \mathbb{P}(S < 0.5). \end{aligned}$$

S is a Gaussian mixture (see section 4.3 for details), which means its c.d.f. is a mixture of Normal c.d.f.'s, making the probability calculations easy.

4 Models

This section describes models which were used for game outcome prediction.

4.1 Baselines

After performing the data pre-processing, we applied several simple models on the transformed data. These simple models served as baselines for deep learning models which are described below.

Our baseline models follow the standard classification set-up. For a given game, the baseline model predicts the class label $y \in \{\texttt{Win}, \texttt{Draw}, \texttt{Lose}\}$ which represents the outcome of the game for the home team. We ask the model to make prediction after each period of the game, so we consider each period of the game as an individual example. Each data example is described by a set of features consisting of:

- **History features**. The history features are computed from past game statistics. For each history statistic feature we compute the difference of the value of the feature for home team and the same value for away team.
- **Period-aggregated features**. The period-aggregated features are computed from aggregated statistics of in-game events which occurred up-to a present moment. Similarly to the history features, each period feature is computed as difference of values for home and away teams.

We use the following models as baselines:

- **Constant**. The constant classifier outputs the most frequent class label $y = \texttt{Win}$ for any input. This classifier provides the simplest baseline and is expected to be outperformed by any sensible model.
- **Goal difference**. This classifier makes prediction based on the current goal difference s only. The goal difference mapped to the class label according the rule: $y = \texttt{Win}$ if $s > 0$, $y = \texttt{Lose}$ if $s < 0$, and $y = \texttt{Draw}$ if $s = 0$. We compare the performance of this baseline to performance of other models in order to make sure that more complex models use other useful signals represented in the data beyond the current goal difference.
- `scikit-learn` **classifiers**. We use the following standard classifiers implemented in `scikit-learn`:
  - **Logistic regression** implemented in `sklearn.linear_model.LogisticRegression`;
  - **Random forest** implemented in `sklearn.ensemble.RandomForestClassifier`;
  - **Gradient boosting** implemented in `sklearn.ensemble.GradientBoostingClassifier`.

4.2 RNN

Recurrent neural networks (RNN) are designed to process sequential data. We implemented an RNN which predicts the outcome of the game given the sequence of events occurred in the game.

Formally, our RNN model operates with the following data:

- f_h — history features vector computed as described in 4.1.
- N — number of events in the game.
- $e_1, e_2, \ldots, e_N$ — sequence of game events. e_i is a feature vector representing i-th event in the game. The feature representation e_i is computed as signed one-hot encoding of the game type (see left column of Table 3). In the signed one-hot vector e_i, all elements are equal to zero except the element representing the current event type. This element holds the value $+1$ is the event is associated with an action of the home team and -1 for the away team event.
- $s_0, s_1, s_2, \ldots, s_N$ — the sequence of goal difference values. s_i is the $\texttt{Home} - \texttt{Away}$ goal difference at the moment after events $e_1, \ldots, e_i$. s_N represents the final foal difference in the game. We denote the initial zero goal difference at the start of the game by $s_0 = 0$.
- $y \in \{\texttt{Win}, \texttt{Draw}, \texttt{Lose}\}$ — target label representing the outcome of the game.

Figure 2 shows the architecture of the RNN model. The model consists of the following components:

- F — history feature transformation network. This network takes the history statistics vector f_h and transforms it into a initial hidden state h_0 of the recurrent module described above: $h_0 = F(f_h)$.
- `GRU` — GRU [4] layer. This recurrent layer processes the input sequence of game events $e_1, \ldots, e_N$. At i-th step, takes the representation of the current event e_i as an input and updates the old value of its hidden state h_{i-1} to compute the new value $h_i = \texttt{GRU}(h_{i-1}, e_i)$.
- P — the game outcome prediction network. This network is trained to predict the game outcome y taking the current hidden state h_i as the input. The final layer of P is a soft-max layer which defines a distribution $q_P(\cdot|h_i)$ over the class label y.
- `Aux` — auxiliary prediction network. This network is trained to perform the auxiliary prediction task. `Aux` takes the current hidden state h_i as an input and outputs an estimate of the current goal difference $\tilde{s}_i = \texttt{Aux}(h_i) \approx s_i$.

We train the model by minimizing the loss function:

$$\sum_{i=0}^{N} [\mathscr{L}_P(h_i, y)] + \lambda \sum_{i=0}^{N} [\mathscr{L}_{\texttt{Aux}}(h_i, s_i)] \to \min_{F, \texttt{GRU}, P, \texttt{Aux}}$$

4.3 MD-RNN

We implemented MD-RNN, a version of mixture density network (MDN) [5].

A mixture density layer outputs parameters π, μ, σ^2 of Gaussian mixture. We add an MDN module which estimates the final score difference s_N using the current hidden state h_i:

$$q_{\texttt{MDN}}(s|h) = \sum_{j=1}^{K} \pi(h)_j \, \mathscr{N}(s; \mu(h)_j, \sigma^2(h)_j).$$

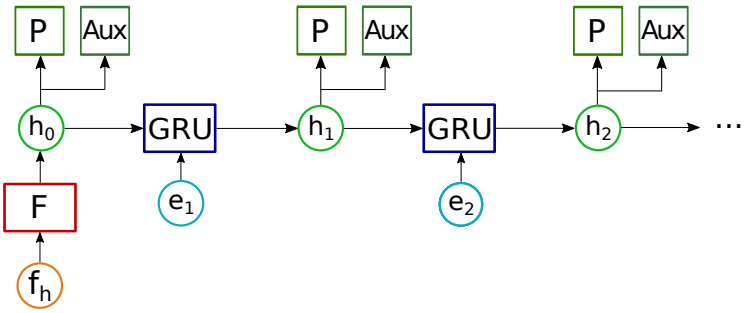

Figure 2: RNN architecture

We train the MDN layer by minimizing the negative log-likelihood loss:

$$\mathscr{L}_{\texttt{MDN}}(h_0,\ldots,h_N,s_N) = \sum_{i=0}^{N} \log q_{\texttt{MDN}}(s_N|h_i).$$

5 Results

	Observed game interval		
Model	0 periods	1 period	2 periods
Constant	49.50	49.50	49.50
Goal difference	–	50.15	68.51
Logistic regression	54.59	63.75	74.81
Random forest	56.37	64.18	74.64
Gradient boosting	56.19	63.86	74.85
RNN	50.69	66.90	77.82
MD-RNN	50.78	67.05	77.68
MD-RNN Augmented	50.78	65.41	77.19

Table 4: Prediction accuracy of game outcome after k periods for the baseline and RNN models.

The numbers in table 4 seem shockingly low, especially the first period ones, but compared to the current machine learning literature on predicting NHL games, the best models are barely able to breach 60% accuracy [6]. As mentioned in the earlier sections, we are not interested in improving this accuracy, but are instead focusing on how the accuracy improves as the game goes on. We decided to choose randomly three different games, one win, draw and loss, for the next analysis. We will first show the effect of different in-game events on the win/draw/lose probabilities (as defined in section 3.2), then show how the score difference distributions change over time, extending Ganguly and Frank's paper [1] to recurrent neural networks, and finally use the addition of the mixed density layer to show a "confidence" graph of the true outcome, as a measure of how well the model is predicting the game outcome and score difference over time. We will compare all of the results from using the regular data set to the augmented one.

5.1 Impact of plays on prediction

As figure 3 shows, the single only important in-game event is unsurprisingly a goal. What is interesting about this result is that shots or hits have close to no impact on the win probability. That is, essentially the model tries to predict the odds of the home team scoring enough goals to win. This would also explain why the pre-game prediction rate is 50%. Since no goals have been observed, even after seeing the recent game history, the model does not have enough information to predict whether the home team can score enough goals in this specific game. Since the data augmentation adds Gaussian noise to the existing features it is not surprising to see that the impact of plays is almost identical for the augmented model.

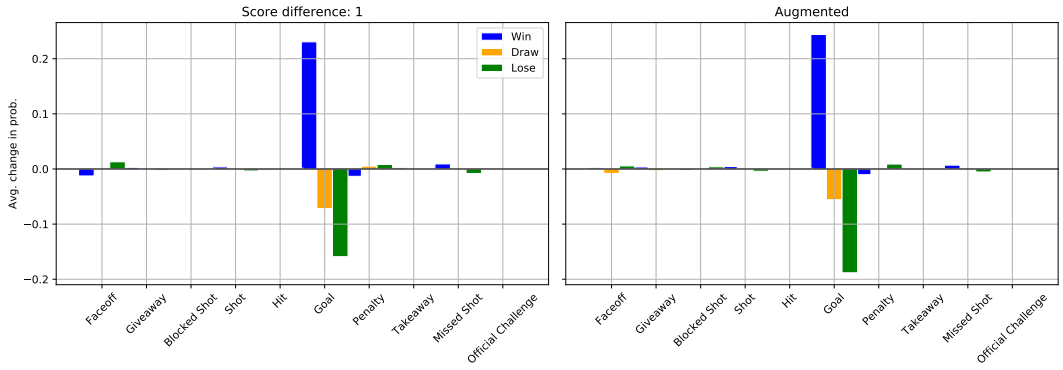

Figure 3: The average change in probability after the specified type of event occurred throughout the game. That, $\mathbb{P}_{\text{after}} - \mathbb{P}_{\text{before}}$. Only one of the three games is shown here, in particular the "win" case, though the effect was similar for all the examples.

5.2 Score difference over time

As mentioned in section 3, we wanted to implement the score difference distribution approach and see whether it gives us some extra information a basic win/draw/lose accuracy cannot.

In figure 4 we highlight three test games (not included in the training data). Each plot is created by stopping the MD-RNN after the given time and plotting the density of the resulting mixture. This allows us to peek inside the MD-RNN and see how it adjusts as the game goes on. In particular, we can see that in the beginning the distribution is quite symmetric, explaining the 50% accuracy before the game starts. Interestingly, even though figure 3 suggests that only goals have an effect on the prediction, the "win" example shows that even the very small probability changes for hits, shots, face-offs aggregate and can cause a distribution change even if no goal was scored. The big changes occur when goals are scored though, shifting the distribution to whichever side scored. Finally, more so than in Ganguly and Frank's NBA example [1], the distribution becomes more and more unimodal as time goes on, indicating that "confidence" over a score difference increases. To visualize this shift in confidence further, we decided to plot the likelihood of the true game outcome over time in figure 5. Interestingly, the confidence of the true outcome does not always increase as the lose example in figure 5 showcases. The reason for this loss of confidence can be inferred from figure 4. Note that midway through the game, home is down by 3 goals and the score distributions are clearly shifted to the left. Then the home team continues to catch up with two goals, making the end result of 0-1 not as likely and thus explaining the lack of confidence. An important thing to keep in mind is that "confidence" here is the likelihood of the true score difference, i.e. not the game outcome, lowering the overall confidence by the fact that score difference is continuous. To gain some more predictability back, we also decided to plot the win/draw/lose probabilities over time of the same games in section 5.3.

First we want to compare the non-augmented with the augmented model and see if there are any noticeable patterns. From figure 3 we see that the effect of goals is greater and all the other small effects are even more reduced in the augmented case. That would imply that the score difference distribution ought to be more sensitive to goals. While they start out the same, as goals are scored, the augmented case becomes unimodal much quicker than its counterpart, supporting the hypothesis that it is more sensitive to goals. What that means is that for instance in the "lose" case, after the first period the augmented case is pretty much unimodal and predicts a loss, as the home team scores, the distribution starts shifting right entirely, whereas the non-augmented one, accounts for the changes by shifting its modes, thereby showing a less abrupt change.

5.3 Win/Draw/Lose probability over time

Because the score outcome is a continuous measure and a distribution plot is hard to translate into concrete outcomes, we also include the win/draw/lose probabilities defined earlier. We believe that combining figures 4 and 6 give the best analysis of a game when used together. On one hand we can use the score difference distributions to see our current belief state and at the same time use the win probability as a concrete outcome measure. As an example, take the "win" case. At time $t = 1.75$, we see that both modes are shifted to the right indicating that the model is very much indicating a home win, the correct outcome. Using figure 6 we see that it does so with probability of about 0.85, which is quite high. At the same time the confidence for the true score outcome is not very high at about 0.15. What this means is that

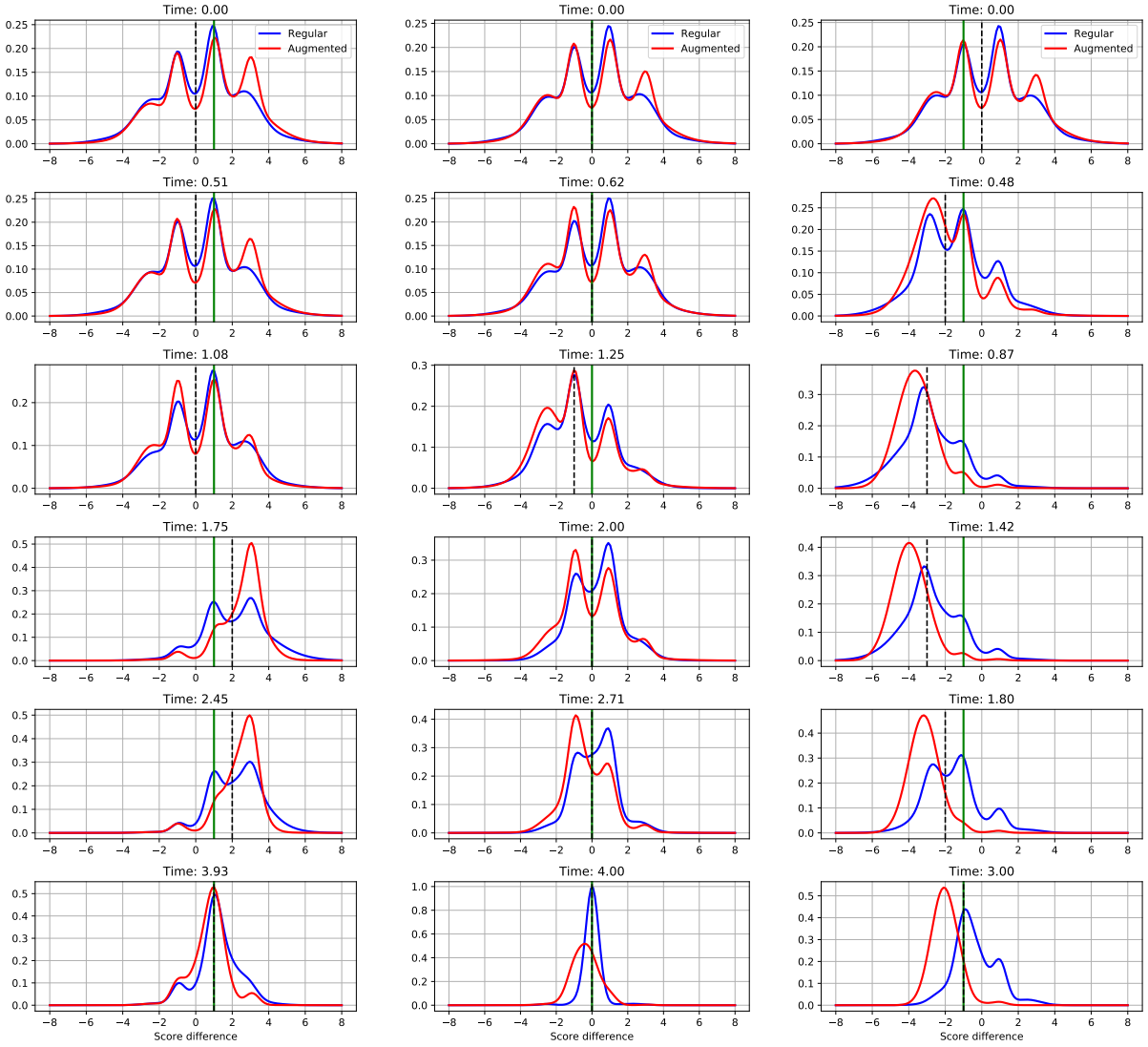

Figure 4: Likelihood as defined by mixture density model returned by MD-RNN at time t. Blue is the model trained on the non-augmented data and red is the model trained on augmented data. The green line indicates the true score difference at the end of the game. The black dashed line represents the current score difference. Figure based on [1].

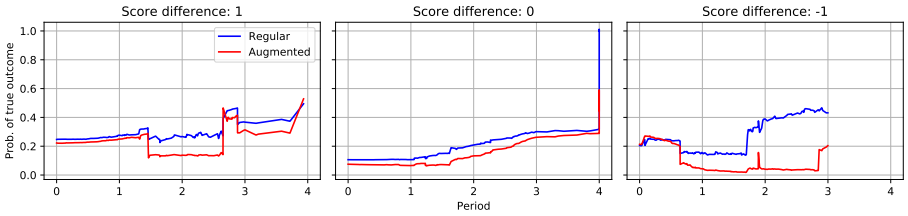

Figure 5: The likelihood of the true score difference over time for all three games.

the model while not great at predicting the score difference, incidentally is good at predicting the game outcome, even though it is trained on the score difference.

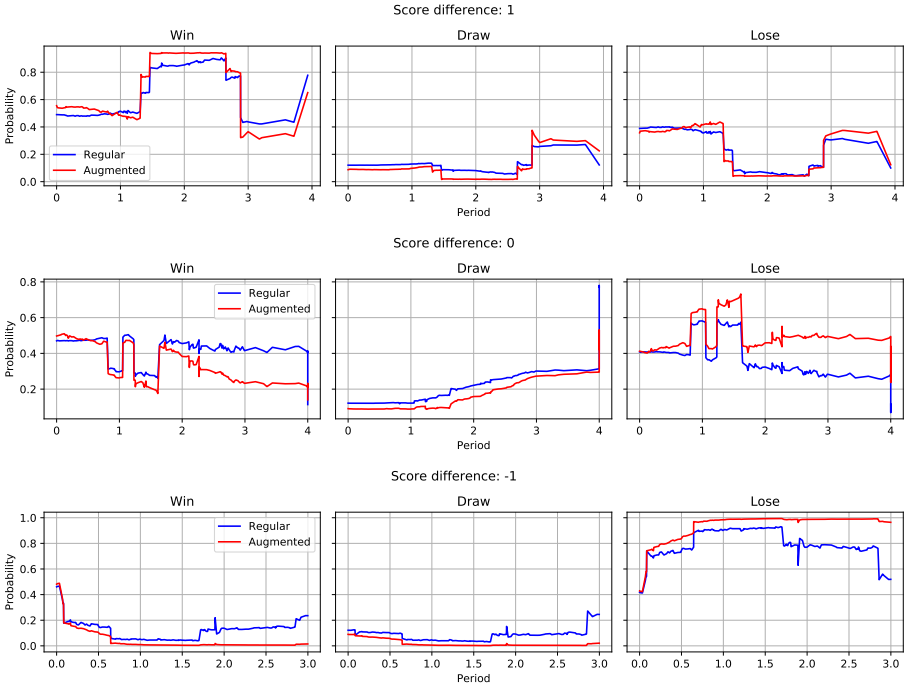

Figure 6: Win/Draw/Lose probabilities over time for the three different games.

Comparing the augmentation it is clear that the probabilities are very similar. But just like the quick convergence to unimodality in figure 4, it follows that it acts as an sort of smoothing effect. So the augmented model seems to be working very well in the "win" and "lose" case, but for a draw it is not as good. That makes sense since we implemented the draw event as an interval around 0, and a unimodal distribution is more likely to over- or underestimate the draw compared to a flatter multimodal (non-augmented) one.

6 Conclusion

Our analysis concludes with the admission that our MD-RNN is not very good at predicting the game outcome before anything has been played, but allows for a much more detailed analysis of the game over time. In particular, training the MD-RNN on the score difference does not drastically reduce the accuracy of the regular RNN trained on the game outcome and provides an easily measurable win probability as well as a good way of visualizing the game. This is useful for mid-game predictions and also helped to show that all the non-goal plays that happen in a game do not seem to contribute as much to the final outcome as we thought in the beginning. Finally, our data augmentation approach did not improve the accuracy of the model and instead ended up making the model more sensitive in the beginning and less so towards the end compared to the more consistent change of the non-augmented approach.
Working on this project made us realize how many features we left out of our actual analysis and we would like to include in future work. These are player attributes currently on the ice at a given time, aggregated player attributes in the team and possibly an ice rink player layout, similar to the NBA approach where they added a feature which estimated the lineup [1]. We also want to include more sophisticated augmentation methods such as an alternative one suggested in [2]. In this method one first creates a list of the k nearest neighbors of a given point x in terms of minimum Euclidean distance of the feature representations of the points. Then for each pair of neighbors x_i and x_j a new point $x'_{i,j}$ can be created where $x'_{i,j} = (x_i - x_j)\lambda + x_j$. This method may be slightly more promising since similar data points are used to create new ones as opposed to a random process to create new points. The hope of this would be to improve the accuracy of our model by having more viable data points to train off of.
Work distribution. Timur Garipov implemented the RNN and MD-RNN models in `PyTorch` and also ran the baseline models using `scikit-learn`. Adam Kaplan implemented all the measures used in the result section as well as all of the plots using `PyTorch`, `matplotlib`. Alexander Lynch modified the models to incorporate the data augmentation technique. All team members participated in the discussion and contributed to the milestones and the final report.

References

[1] Sujoy Ganguly and Nathan Frank. The problem with win probability. In *Proceedings of the 12th MIT Sloan Sports Analytics Conference. Boston*, 2018.

[2] Terrance DeVries and Graham W Taylor. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017.

[3] `NHL Game Data`. https://www.kaggle.com/martinellis/nhl-game-data.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] Christopher M Bishop. Mixture density networks. 1994.

[6] Gianni Pischedda. Predicting nhl match outcomes with ml models. *International Journal of Computer Applications*, 101(9), 2014.