Sequential Deep Learning Approach for Predicting the Value of NBA Players on New Teams

Justin Kennedy
Department of Data Science
Columbia University
e-mail: jak2294@columbia.edu

Abstract— **This study uses an adaptive sequential deep learning framework to predict the value NBA players would add if hypothetically traded or signed to new teams. The approach utilizes both individual and team statistics to identify a player's fit within a team's offensive and defensive schemes. Historical instances of players switching teams are sequentially trained in the presented algorithm cases. In both cases, the algorithm's performance was tested on the five hundred most recent player team switches and proved to effectively predict the number of wins players contributed to on their new teams.**

Keywords- NBA player prediction; sequential learning; adaptive deep learning networks;

I. INTRODUCTION

Every year general managers and coaches around the NBA begin the process of looking for new talent to improve their league standing. Generally, this is done through signings in free agency or through trades with other teams around the league. In the NBA, teams can sign players from the start of free agency July 1st to the end of the regular season in mid-April, restrictions permitting, while trades can be made from early July to the trade deadline in early February.

Very often, teams pay big for players who performed well on previous teams in expectation that this level of performance will transfer over. However, this is not always the case. In fact, several examples in recent history highlight this discrepancy. Dwight Howard and Kevin Love are two recent players whose performance dipped heavily after changing teams while players like Victor Oladipo and James Harden saw their talents, and subsequently their league-wide recognition, blossom on new teams. Off of this, we're looking to account for the fact that given some team's current schemes, some players can be inserted into these schemes and add a lot of value, whereas others may have trouble carrying over the same production in the new scheme; here, we are ignoring variables outside of their performance on the court. As an example, if you were the Golden State Warriors and could hypothetically add LeBron James, you likely would not expect him to add as much value to the team as compared to how much he currently adds to the Lakers given the fact you already have 5 all-stars taking up playing time that the team's plays currently revolve around. In this sense, a less well-known player could hypothetically add more value to the Warriors in the form of wins to their team than LeBron James.

The goal of this project is not to assess the value of a target player individually, but to instead assess his value as his 'fit' with a given team. Thus, the problem to solve becomes how we can map a player's performance on one team to his performance on another team through evaluation of both team and individual variables.

Additionally, we are looking to design an algorithm that updates accordingly with time as the style of play in the NBA changes. In the 1980's, the short-range jump shot was very common for teams to revolve their offense around. In today's NBA, many teams have adapted to a game plan based heavily on a high volume of three point shots. This recent push has put so-called '3 and D players' (players who efficiently shoot three points along with great defense) in high demand, as teams look to utilize these players heavily in their offense.

Here I propose the use of a sequential modelling approach utilizing a deep learning regression network that adapts its architecture for each given training and validation set pairing to predict the value of players on new teams. Specifically, by evaluating a subset of players who have changed teams historically, I look to produce an algorithm that updates its parameter weights after each player case, sequentially in time, and thus adapts to the changes in style of play around the NBA. To evaluate the efficacy of the algorithm, I will test on player cases directly proceeding different-sized training datasets and compare the player's predicted added value to his actual added value in terms of Win Shares per 48 minutes. This metric is discussed in more depth in Section (III).

II. RELATED WORKS

Similar to the proposed approach, the authors in [1] used both individual and team independent variables and win shares as the target variable; however, they sought to predict the win shares of a player game to game. The system in [1] utilizes a linear mixed model approach to predicting the performance of NBA players. While the proposed algorithm could be modified to predict player performance in this setting, the main aim is to predict a player's value transfer team to team on a season by season basis. Additionally, the approach in [1] doesn't include a 'fit' component of a player

with his team to predict performance as does the proposed method. FiveThirtyEight, a website well known for publishing intriguing sports predictions, also has a rating, called a CARMELO rating. This rating seeks to predict the value NBA players add to their team during a given season in terms of plus-minus compared to replacement, but instead does this through calculating similarity values between a given player and other players with similar physical characteristics and individual statistics without incorporating a team fit component.

III. System Overview

The datasets used in this study were from Kaggle and BasketballReference.com. The Kaggle player statistics dataset included individual statistics of NBA players each season since 1980. The 'Season_Stats' dataset included variables such as points per game, minutes played, player efficiency rating, win shares, shooting efficiency among others. The overall dimensions of the dataset are (24.7k x 53). Three datasets were used from BasketballReference.com, specifically ones titled 'Team Per Game Stats', 'Opponent Per Game Stats', and 'Miscellaneous Stats'. These tables were extracted from each season since 1980. These datasets included team variables such as three point attempt rate, two pointers allowed per game, turnover percentage, and pace of play. The dimensions of these datasets for each year were (31x25), (31x25), (31x27) respectively (39 years equals each table dimension x 39).

The proposed system encompasses preprocessing, feature-engineering, and model creation stages. To preprocess the player stats dataset, each instance of a player switching teams was detected and extracted. The player's individual stats on the old team and the new team were appended in the same row in a new dataframe (Note: Many players have switched teams several times in their careers since 1980; each instance was set as a new row in the formed dataframe). In this new dataframe of team switches, each player's discrete stat (ex: points, rebounds, assists) was put on a per minute basis by dividing each stat by the player's minutes played. This transformation was done to put players with different amounts of playing time on the same scale in terms of productivity; otherwise, the predicted added value of players with lower levels of minutes played would be significantly less than players with more minutes and the model would have little predictive power in cases where players received more minutes on the new team.

Before merging this player switch dataframe with the accompanying team statistics of their old teams and new teams, the data formats of the two sets had to be standardized. For the player switch dataframe, the team name abbreviations were converted to the actual team names (PHO → Phoenix Suns) to agree with the team statistics dataset. Additionally, many variables in the team dataset had to be renamed due to conflict with the variable names in the player switches dataframe. Teams who have had name switches since 1980 had to also be renamed to a common standard to prevent confusion between the player and team datasets; teams in particular that had to be adjusted accordingly were the New Orleans Hornets/Charlottle Bobcats/Charlotte Hornets (the team had several name changes in the 2000s). All three team statistics dataset were then merged together to put each team's statistics for each year side by side.

Finally, for each row in the player switches dataframe, the team statistics combined dataframe was looped through to find the stats of the player's old team and new team of the particular year of the player's switch. These team statistics were appended accordingly row by row in the player switches dataframe.

For feature selection in the resulting player/team statistic merged dataframe, redundant variables were first taken out. For example, since 3P%=3P made / 3P Attempted, only 2 out of three of these variables is needed. Next, variables irrelevant to the problem were dropped. These excluded variables included most of the player statistics on the second team besides minutes played and usage % (usage % = percentage of offensive plays run by the team involving the given player). This is because we are looking to predict a player's value added to the new team without actually knowing how he performed on that team. However, I did keep the minutes played and USG% variables, as a team can hypothetically use the proposed algorithm to see how a player would perform given varying levels of MP and USG% when integrating into the new team's system. Finally, the team statistics of the old team and new team for each player were each converted into one representative variable. For each percentage statistic, the resulting variable became the difference percentage-wise between the stat on team 2 vs team 1. For discrete variables, the percentage difference was taken between team 1 and team 2. As an example, if a player's old team averaged 40 rebounds per game and his new team averaged 50 rebounds per game, the resulting rebounds per game variable to replace these in the dataframe would be +25%. The final dimensions of the player/team statistics dataframe of player switches after feature selection and modification came out to (5400x105). The target variable used in the study to estimate a player's value added on a new team was Win Shares / 48 minutes, which estimates the number of 'wins' a player contributes to his team, aggregating both offensive and defensive production on a per 48 minute basis.

The variables in the player/team statistics player switches dataframe were used as feature inputs into a deep learning regression network. The target variable to predict by the regression, as stated above, was the Win Shares / 48 minute variable.

IV. ALGORITHM

To design the architectures of the deep learning networks for the presented training set cases, the hyperas package was used to automatically determine the optimal models, given the respective training and testing sets. The hyperas package 'optim' function takes as input a framework of parameters to be optimized and outputs the parameter values that produce the model with the least validation loss in the testing set after a given number of epochs (in this case, validation loss is calculated using mean average error). In the proposed approach, the variables optimized were number of layers, number of units per layer, dropout percentage, activation types, learning rate, and momentum rate.

In this sense, the hyperas package functions are being used to adaptively determine the optimal neural network frameworks for given training sets by iterating through different combinations of the chosen variables to optimize. Given a stream of different training sets, the algorithm can determine the ideal architectures for each based off of the validation loss each iteration produces. By giving the 'optim' function a large framework (twelve layers) to potentially build up to, in finding the model with the least validation loss, the algorithm is essentially using a top-down approach, where at maximum size the model can be twelve layers, but if a more compact representation produces the optimal solution then that representation is outputted. However, it should be noted that this algorithm is presented as a sequential modelling approach as opposed to specifically an online learning approach due to the fact that this neural network optimization process is solely designed to optimize the validation loss without regards to efficiency. In the current maximum twelve layer format, the optimization process takes approximately five to ten minutes to run on a Nvidia Tesla P100 and therefore would likely not be currently feasible in a constant data streaming setting.

The main case presented included using the first 4800 team switches since 1980 for training and the most recent 500 team switches for testing. This represents the extreme end of number of training cases to present to the model which updates its parameter weights by evaluating each player switch one-by-one. In this case, the deep learning network was created and implemented using the Keras package in python. The network utilized a stochastic gradient descent optimizer with a batch size of one to envelope the desired sequential learning characteristics. In addition, the momentum feature of the stochastic gradient descent optimizer was utilized to further promote the desired effect of the model to value more recent player switches more so than ones further in the past by accelerating the gradient vector in the correct direction for each player evaluation.

The second test cast presented utilizes the 500 most recent players switches for testing, similar to the exhaustive approach, but instead trains on a much smaller subset of the previous 1000 player switches. The reason for having this second test case is to observe if the prediction results for a much smaller look-back period are maintained and to see how these results compare with other approaches.

As in both the main and second testing cases, the property is maintained that the training examples directly precede the testing examples. This is because the algorithm is designed to predict future data points using the most recent cases for training and adapting accordingly. If as in shuffled batch testing, 'future' data points were used in training, this would contradict the desired sequential learning property of being able to observe trends over time and predict future player values using the most "up-to-date" model.

The produced optimal architecture for the whole training set case by the hyperas package framework is shown in Figure 2.

```
def get_space():
    return {
        'Dense': hp.choice('Dense', [16,32, 64]),
        'Activation': hp.choice('Activation', ['relu', 'sigmoid', 'tanh']),
        'Dropout': hp.uniform('Dropout', 0, 1),
        'Dense_1': hp.choice('Dense_1', [16,32,64,256]),
        'Activation_1': hp.choice('Activation_1', ['relu', 'sigmoid', 'tanh']),
        'Dropout_1': hp.uniform('Dropout_1', 0, 1),
        'Dropout_2': hp.choice('Dropout_2', ['two', 'three']),
        'Dense_2': hp.choice('Dense_2', [16,32,64,256, 512]),
        'Activation_2': hp.choice('Activation_2', ['relu', 'sigmoid', 'tanh']),
        'Activation_3': hp.choice('Activation_3', ['two', 'three', 'four']),
        'Dense_3': hp.choice('Dense_3', [16,32,64,256]),
        'Activation_4': hp.choice('Activation_4', ['relu', 'sigmoid', 'tanh']),
        'Dropout_3': hp.uniform('Dropout_3', 0, 1),
        'Dense_4': hp.choice('Dense_4', [16,32,64,256]),
        'Activation_5': hp.choice('Activation_5', ['relu', 'sigmoid', 'tanh']),
        'Activation_6': hp.choice('Activation_6', ['two', 'three', 'four', 'five']),
        'Dense_5': hp.choice('Dense_5', [16,32,64,256]),
        'Activation_7': hp.choice('Activation_7', ['relu', 'sigmoid', 'tanh']),
        'Dropout_4': hp.uniform('Dropout_4', 0, 1),
        'Dense_6': hp.choice('Dense_6', [16,32,64,256]),
        'Activation_8': hp.choice('Activation_8', ['relu', 'sigmoid', 'tanh']),
        'Dropout_5': hp.uniform('Dropout_5', 0, 1),
        'Dense_7': hp.choice('Dense_7', [16,32,64,256]),
        'Activation_9': hp.choice('Activation_9', ['relu', 'sigmoid', 'tanh']),
        'Activation_10': hp.choice('Activation_10', ['two', 'three', 'four', 'five', 'six']),
        'Dense_8': hp.choice('Dense_8', [16,32,64,256]),
        'Activation_11': hp.choice('Activation_11', ['relu', 'sigmoid', 'tanh']),
        'Dropout_6': hp.uniform('Dropout_6', 0, 1),
        'Dense_9': hp.choice('Dense_9', [16,32,64,256]),
        'Activation_12': hp.choice('Activation_12', ['relu', 'sigmoid', 'tanh']),
        'Dropout_7': hp.uniform('Dropout_7', 0, 1),
        'Dense_10': hp.choice('Dense_10', [16,32,64,256]),
        'Activation_13': hp.choice('Activation_13', ['relu', 'sigmoid', 'tanh']),
        'Dropout_8': hp.uniform('Dropout_8', 0, 1),
        'Dense_11': hp.choice('Dense_11', [16,32,64,256]),
        'Activation_14': hp.choice('Activation_14', ['relu', 'sigmoid', 'tanh']),
        'Dropout_9': hp.uniform('Dropout_9', 0, 1),
        'lr': hp.choice('lr', [0.0001,0.00001, 0.000001]),
        'momentum': hp.choice('momentum', [0.6,0.7,0.8,0.9]),
    }
```

Figure 1: The above framework is defined by the hyperas package based on the input selection of which variables to alter to optimize the validation loss for the best model. Each line in the above section with 'hp.choice' indicates values to be chosen to optimize the overall model. The hyperas optim function runs through every possible combination of the choices in this optimization process. In the lines with a choice relating to numbers in string form, ex "two, three, four", the networks determining whether two, three, or four layers would create the model with the least validation loss. The code used to produce the above framework is shown in Figure 4.

```
Evalutation of best performing model:
500/500 [==============================] - 0s 89us/step
0.06691111105680465
Best performing model chosen hyper-parameters:
{'Activation_8': 1, 'Dense_10': 1, 'Activation_9': 0, 'Dense_8': 1, 'lr': 0, 'Dropout_1': 0.3720818863636862, 'Activation_
6': 2, 'momentum': 3, 'Activation_14': 1, 'Activation_7': 2, 'Dense_2': 0, 'Dense_6': 1, 'Dense_3': 0, 'Dense_1': 2, 'Dense_
9': 2, 'Activation_5': 2, 'Dropout_3': 0.7937157362201431, 'Dense_4': 1, 'Dense_11': 3, 'Dense_5': 3, 'Dropout_5': 0.953273911
36720357, 'Dropout_7': 0.587606728324542, 'Dropout_8': 0.3746350041674067, 'Dropout_2': 0, 'Activation': 0, 'Dropout_4': 0.83
64280725819752, 'Activation_12': 1, 'Activation_13': 0, 'Activation_1': 1, 'Activation_2': 1, 'Dropout_6': 0.13749544245386258
8, 'Dense': 0, 'Activation_10': 1, 'Activation_4': 2, 'Activation_11': 0, 'Dense_7': 2, 'Dropout_9': 0.5838315653286106, 'Act
ivation_3': 1, 'Dropout': 0.4350595881468875}
```

Figure 2: The outputted best model results using the hyperas package for optimizing the validation loss on the full training set size case. The dictionary output at the bottom represents the optimized values to fit the parameters of the Figure 1 'choice' structure.

V. Software Package Description

The accompanying code is able to automatically preprocess inputted data in the structure of the team and player statistics datasets above and convert them into one dataframe of the desired format. This is largely done using pandas operations over the respective dataframes. The following code, for instance, converts the inputted player statistics dataset to a new dataframe displaying each team switch by every player over the course of their careers and then puts all the discrete variables in per minute terms. The user would simply input their own csv at the top and run the proceeding code.

```
df = pd.read_csv('Seasons_Stats.csv')
df2= df.sort_values(by=['Player', 'Year'])
df2=df2.dropna(subset=['Player'])
df2 = df2[df.Tm != "TOT"]

playerMoveDS = pd.DataFrame(columns=['Unnamed: 0', 'Year', 'Player', 'Pos', 'Age', 'Tm', 'G', 'GS',
       'MP', 'PER', 'TS%', '3PAr', 'FTr', 'ORB%', 'DRB%', 'TRB%', 'AST%',
       'STL%', 'BLK%', 'TOV%', 'USG%', 'blank', 'OWS', 'DWS', 'WS',
       'WS/48', 'blank2', 'OBPM', 'DBPM', 'BPM', 'VORP', 'FG', 'FGA',
       'FG%', '3P', '3PA', '3P%', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA',
       'FT%', 'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF',
       'PTS', 'Unnamed: 01', 'Year1', 'Player1', 'Pos1', 'Age1', 'Tm1', 'G1', 'GS1',
       'MP1', 'PER1', 'TS%1', '3PAr1', 'FTr1', 'ORB%1', 'DRB%1', 'TRB%1', 'AST%1',
       'STL%1', 'BLK%1', 'TOV%1', 'USG%1', 'blank1', 'OWS1', 'DWS1', 'WS1',
       'WS/481', 'blank21', 'OBPM1', 'DBPM1', 'BPM1', 'VORP1', 'FG1', 'FGA1',
       'FG%1', '3P1', '3PA1', '3P%1', '2P1', '2PA1', '2P%1', 'eFG%1', 'FT1', 'FTA1',
       'FT%1', 'ORB1', 'DRB1', 'TRB1', 'AST1', 'STL1', 'BLK1', 'TOV1', 'PF1',
       'PTS1'])
npdf2=df2.values
j=0
for i in range(npdf2.shape[0]):
    if npdf2[i][2]==npdf2[i-1][2]:
        if npdf2[i][5]!=npdf2[i-1][5]:
            a=np.append(npdf2[i-1],npdf2[i])
            playerMoveDS.loc[j]=a
            j+=1

colstostd=['OBPM', 'DBPM', 'BPM','FG', 'FGA',
       '3P', '3PA', '2P', '2PA', 'FT', 'FTA',
       'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF',
       'PTS', 'MP1','OBPM1', 'DBPM1', 'BPM1','FG1', 'FGA1',
       '3P1', '3PA1', '2P1', '2PA1', 'FT1', 'FTA1',
       'ORB1', 'DRB1', 'TRB1', 'AST1', 'STL1', 'BLK1', 'TOV1', 'PF1',
       'PTS1', 'DWS', 'OWS']

playerMoveDS.iloc[5000]

for i in colstostd:
    playerMoveDS[i]=playerMoveDS[i]/playerMoveDS['MP']
```

Figure 3: An example of preprocessing code automation for the player statistics dataset. The code puts all historical player team switches in a new dataframe and standardizes all the discrete features.

After preprocessing, the provided software prepares the training set of specified size for input into the model creator through standardization of features. A create_model function is provided to then output the optimal keras neural network structure using the hyperas package on the given training set case.

```
def create_model(train_data, train_ws, test_data, test_ws):

    model = Sequential()
    model.add(Dense({{choice([16,32, 64])}}, input_shape=(train_data.shape[1],)))
    model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
    model.add(Dropout({{uniform(0, 1)}}))
    model.add(Dense({{choice([16,32,64,256])}}))
    model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
    model.add(Dropout({{uniform(0, 1)}}))

    if {{choice(['two', 'three'])}} == 'three':
        model.add(Dense({{choice([16,32,64,256, 512])}}))
        model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))

    if {{choice(['two', 'three', 'four'])}} == 'four':
        model.add(Dense({{choice([16,32,64,256])}}))
        model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
        model.add(Dropout({{uniform(0, 1)}}))
        model.add(Dense({{choice([16,32,64,256])}}))
        model.add(Activation({{choice(['relu', 'sigmoid', 'tanh'])}}))
```

Figure 4: The beginning of the hyperas optimization function is shown (for visibility purposes, only the first four layer choices shown; actual model goes twelve layers). This input code, when ran, produces the framework shown in Figure 1 and outputs the best model parameters as shown in Figure 2.

```
    model.add(Dense(1)) #Since its a regression network,
                        #output layer has 1 unit (not customizable)


    model.compile(loss='mae',optimizer=optimizers.SGD(
            lr={{choice([0.01,0.001,0.0001, 0.00001])}},
            momentum={{choice([0.2,0.4,0.6, 0.8])}}))

    result = model.fit(train_data, train_ws,
                       validation_data=(test_data,test_ws),
                       batch_size=1,
                       epochs=20,
                       verbose=2)

    #get the highest validation accuracy of the training epochs
    validation_loss = np.amin(result.history['val_loss'])

    print('Best validation acc of epoch:', validation_loss)

    return {'loss': validation_loss, 'status': STATUS_OK, 'model': model}


best_run, best_model = optim.minimize(model=create_model,
                                      data=data,
                                      algo=tpe.suggest,
                                      max_evals=5,
                                      trials=Trials(),
                                      notebook_name='simple_notebook')
```

Figure 5: The last lines in the create_model function shown in Figure 4. Additional parameter optimization is shown for learning and momentum rates. 'Best_model' returns the architecture of the best performing model for the given training and testing sets.

VI. Experiment Results

Validation loss in the deep learning network was measured by mean average error. This loss was used as opposed to mean squared error due to the fact the WS/48 target variable is generally between 0 and 1. The performance of the algorithm in both training set cases is compared to a non-deep learning approach using the sklearn package SGD regressor. The results of the two algorithm approaches are shown in Table 1.

Approach	MAE Training Size: 4800	MAE Training Size: 1000
Proposed	0.066	0.066
Sklearn SGD	0.081	0.088

Table 1: The resulting mean average errors between the actual testing data and the predicted data using the Proposed and Sklearn SGD approaches on training sets of size 4800 and 1000 respectively. In both cases, the testing data was the 500 most recent player team switches.

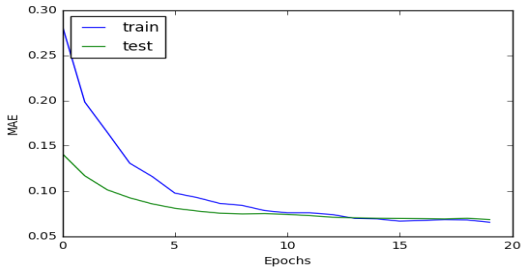

Figure 6: The training and testing loss over twenty epochs of the optimized deep learning network on the entire training set case (n=4800).

	Name	Year	Team1	Team2	ActualWS/48	PredWS/48
5879	Zaza Pachulia	2017	Dallas Mavericks	Golden State Warriors	0.18	0.18
1943	Ersan Ilyasova	2017	Orlando Magic	Oklahoma City Thunder	0.01	0.07
4186	Nene Hilario	2017	Washington Wizards	Houston Rockets	0.16	0.14
2051	George Hill	2017	Indiana Pacers	Utah Jazz	0.18	0.18
2030	Gary Neal	2017	Washington Wizards	Atlanta Hawks	-0.19	-0.05
550	Boban Marjanovic	2017	San Antonio Spurs	Detroit Pistons	0.28	0.34

Figure 7: Actual Win Shares per 48 minutes on Team 2 versus Predicted Win Shares per 48 minutes on Team 2 for a small subset of players who switched teams in 2017.

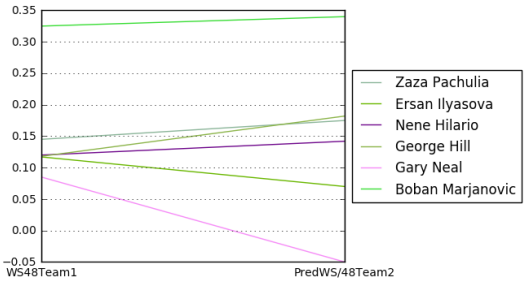

Figure 8: WS/48 on Team 1 versus Predicted WS/48 on Team 2 for a small subset of players who switched teams in 2017. Note: this differs from the presented statistics in Figure 7 as Figure 7 compares the Actual WS/48 on Team 2 vs the Predicted WS/48 on Team 2.

VII. Conclusion

The proposed sequential deep learning approach was shown to effectively predict the Win Shares / 48 minutes of NBA players on new teams. As shown in Table 1, the proposed framework proved to outperform the sklearn SGD regressor approach in both the entire training set and smaller set cases. In fact, the model's performance did not deteriorate at all when the training set size was reduced by almost a factor of five. For future work, I am interested in comparing the proposed approach to batch-learning methods, specifically to see if the effect of changing NBA 'generations' of play is significant. If so, I would expect the sequential modelling approach to generally outperform the batch-learning approaches.

Even though the proposed algorithm was tested on actual player team switches historically, its applications can extend to hypothetical team switches of every player in the league from the perspective from a given team. In this sense, a general manager could run the algorithm on every player on every team (as well as players in free agency) to see how much value each player is predicted to add to his team in terms of win contributions. This would also aid in how to value players in money terms for trade and signing purposes. If a player was predicted to add a lot of value to a team, a general manager may be more inclined to offer more in a trade.

For testing purposes, only two cases were tested: the entire training set and the most recent 1000 training cases (before the test set sequentially). However, in most of its real-life applications, training on the entire data set would be unnecessary, as the algorithm is designed to update the model's parameters case by case and generally one is interested in the most recent trending. With this, the algorithm could just as easily be tested on arbitrarily-sized training sets using the hyperas package functions to adaptively change the depth of the network.

In theory, you would not have to change the architecture of the optimized deep learning network for similarly-sized look-back periods. Ideally, the performance between network architectures for similar sized training sets would be comparable. In fact, the architecture of the proposed algorithm for the entire training set case had comparable results to the second training set architecture for the 1000 sized training set. This shows a potential transferability of effectiveness of similar architectures on different-sized training set sizes. However, due to the risks of overfitting and a vanishing gradient, it would not be recommended to utilize the full network architecture on very small training sets.

In addition, the proposed algorithm can be easily modified to adjust daily during the NBA regular season as new game statistics for players are inputted into the model every day. In this way, the tool can be used to track predicted player value to a given team day-to-day during the regular season. Since the quickest NBA game data can accumulate is day to day (games occur at most once a day), the proposed algorithm would work effectively to produce a model with minimal validation loss in this setting (as of now worst case run-time for the algorithm on largest possible training set is ten minutes). However, if one wanted to modify the approach to predict performance second by second in a game, to more of a customary online learning format, a more efficient algorithm in terms of time complexity would have to be used (or simply better hardware).

An alternative approach to better optimize implementation of the algorithm on vastly different training set sizes in efficient time is to create an adaptive deep learning framework for different sets of training data in an online learning setting as described in [2]. To accomplish this, the system in [2] utilizes a bottom-up approach, starting with a shallow network and growing to a deeper one as necessary. Another potential solution to this problem to be pursued in future work is presented in [3]. Contrasting with the approach in [2], the system in [3] utilizes a sparsity regularization method to start from a large network and reduce in size as necessary. The results in [3] show that the method is able to reduce network sizes and improve performance effectively.

ACKNOWLEDGMENT

I WOULD LIKE TO THANK PROFESSOR CHUNG-YUNG LIN AS WELL AS THE TEACHING ASSISTANTS OF BIG DATA ANALYTICS (EECS E6893): VISHAL ANAND, ZIXI HUANG, YANBEI PANG, PRATYUS PATI, ZEKUN GONG, AND CHAO-YANG LO FOR THEIR HELPFUL FEEDBACK THROUHOUT THE COURSE OF THE PROJECT.

APPENDIX

REFERENCES

[1] M. Casals and J. Martinez, "Modelling player performance in pasketball through mixed models," International Journal of Performance Analytics in Sports, April 2013, pp. 64-82.

[2] D. Sahoo, J. Lu, Q. Pham, and S. Hoi, "Online deep learning: learning deep neural networks on the fly," CoRR, November 2017.

[3] J. Yoon and S. Hwang, "Combined group and exclusive sparsity for deep neural networks," ICML, 2017.