Deep Factor Graphs for Bayesian Prediction of High-Dimensional Games

Alon Daks

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-151
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-151.html

August 16, 2017

Acknowledgement

Thanks to my advisor, Laurent El Ghaoui, for his support, advice, and encouragement while an undergraduate and masters student at Berkeley and to Lisa Goldberg for her collaboration in various statistical projects related to basketball analytics. Thanks to Nishant Desai, who was my closest collaborator and classmate from the very beginning of college through our 5th Year MS. We both lived in Clark Kerr Building 7 as freshman and met during our first week at Berkeley. Much of the work in this thesis is the result of many great conversations we had together. Thanks to Aidan Clark for his collaboration as a research partner and advice for various components of this project. Finally, I'm especially thankful for my parents for their continued encouragement and support throughout my education.

Deep Factor Graphs for Bayesian Prediction of High-Dimensional Games

by

Alon Daks

A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Laurent El Ghaoui, Chair
Professor Lisa Goldberg

Spring 2017

Deep Factor Graphs for Bayesian Prediction of High-Dimensional Games

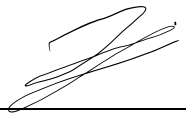
by Alon Daks

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Laurent El Ghaoui
Research Advisor

Aug. 9, 2017

(Date)

* * * * * * *

Professor Lisa Goldberg
Second Reader

11 August 2017

(Date)

Acknowledgments

I would like to thank my advisor, Laurent El Ghaoui, for his support, advice, and encouragement during my time as an undergraduate and masters student at Berkeley as well as Lisa Goldberg for her collaboration in various statistical projects, including this one, related to basketball analytics. I am also thankful to Nishant Desai, who was my closest collaborator and classmate from the very beginning of college through our 5th Year MS. We both lived in Clark Kerr Building 7 as freshman and met during our first week at Berkeley. Much of the work in this thesis is the result of many great conversations we had together. Thanks to Aidan Clark for his collaboration as a research partner and advice for various components of this project. Finally, I am especially thankful for my parents for their continued encouragement and support throughout my education.

Deep Factor Graphs for Bayesian Prediction of High-Dimensional Games

Alon Daks
Computer Science Division
University of California, Berkeley
alon.daks@berkeley.edu

Abstract

This paper offers an extension to TrueSkill, a Bayesian method for ranking players and predicting outcomes of multiplayer games, for cases where a game is high-dimensional. TrueSkill was originally developed by Microsoft Research to rank and match XBox Live players, but offers a general method for inferring player skill based almost exclusively on the win-loss outcome of a match. Although such a method works well for relatively simple games like Halo, the framework is limited in its ability to incorporate information-rich features — often called boxscores — commonly used to describe high dimensional games, such as basketball. Our work extends TrueSkill for these types of games by reformulating its underlying graphical model as the internal dynamics of a recurrent neural network cell in addition to using neural networks as expressive function approximators to map between high-dimesional boxscores and a player's weight when conducting TrueSkill updates. Experimental results on NBA data shows that our method improves upon the original TrueSkill algorithm for predicting the outcome of basketball games.

1 Introduction

In recent years, deep learning has played a central role in achieving state-of-the-art accuracy for many supervised learning problems across several key disciplines of artificial intelligence, namely computer vision, natural language processing and audio recognition. Since neural networks constitute highly expressive function approximators, they have an ability to learn features from high-dimensional datasets that are otherwise unfeasible for humans to craft by hand. Although this ability to automatically learn features from data is instrumental to deep learning's success, it is often the source of its greatest criticism: the decisions made by neural networks are obscured by layers of complex non-linearities and are therefore hard to interpret. On the contrary, probabilistic graphical models are usually easier to interpret since they break down inference problems by expressing joint distributions in terms of conditional independence relationships between sets of random variables. By explicitly defining relationships between component random variables, a graphical model is typically more constrained in the family of functions it can represent, and therefore limited in its ability to learn features. This work proposes an extension to TrueSkill, a graphical model for ranking players and predicting outcomes of multiplayer games, by using neural networks as function approximators to map between high-dimensional boxscores and input weights used during TrueSkill updates. By overlaying deep components on top of the original TrueSkill algorithm, we are able to improve prediction accuracy for games by incorporating more information specific to individual players' performances.

TrueSkill was originally developed by Microsoft Research as a Bayesian method for a) ranking XBox live players and b) generating competitive online matches. Although it was motivated in the video game domain, the algorithm is general to any multiplayer competitive activity. It was offered as an improvement over the more classical Elo rating system, which was first proposed in 1959 by Arpad

Elo as a statistical method for ranking Chess players. Whereas Elo based systems model entities at the team level, TrueSkill based systems further break teams down by modeling individual players. TrueSkill style systems therefore lend themselves to modeling players and teams in professional sports leagues since the performance of a team mostly reflects its roster of currently active players rather than the institution or franchise which it represents. In other words, from a statistical perspective it is much more important to have an understanding of who is playing on a team like the Golden State Warriors rather than merely constructing an understanding of how the Warriors team played in the past. In this paper, we closely examine TrueSkill, and our proposed extension, through the context of evaluating and predicting NBA games. We chose professional basketball due to its popularity and high number of regular season games, which provide for an abundance of training and test data.

This paper is organized as follows. Section 2 presents an overview of the original TrueSkill algorithm, section 3 shows have to reformulate TrueSkill as a recurrent neural network, section 4 extends TrueSkill for high-dimensional games, section 5 discusses our NBA dataset, section 6 outlines the experimental approach, section 7 presents results and finally section 8 provides concluding remarks.

2 Vanilla TrueSkill

The TrueSkill algorithm models posterior distributions over skill levels for players in a game. *Skill* here is an abstract measure, expressed as a random variable, of a player's ability for the particular game in question. For a given match, a player's *performance* is modeled as a random variable centered at the player's skill. Performance, again, is an abstract measure of the player's contribution to his or her team performance in a match. *Team performance* is a weighted sum of a team's players' performances. Finally, the model compares team performances and assigns probabilities to pairs of outcomes.

More formally, the TrueSkill model describes a set of random variables arranged in a factor graph [3]. We will describe the graph here, and Figure 1 provides a visualization. In our setting, we have two teams, each with m players. For each player i, there is a skill node s_i. Each s_i is attached to a factor representing its prior, which is distributed $\mathcal{N}(\mu_i, \sigma_i^2)$ for some mean and variance to be inferred.

For each skill node, there is a performance node, p_i, which is distributed as a Gaussian centered at s_i:

$$p_i \sim \mathcal{N}(s_i, \beta^2)$$

where β is a model parameter that is the same for every player.

The team performance node, t is a weighted sum of performances:

$$t \sim \sum_i w_i p_i$$

where the weights w_i are also model parameters. In the factor graph representation, this relation is represented as a single factor connected to all the p nodes and to the t node, where the factor function is given by:

$$\mathbb{I}\left(t = \sum_i w_i p_i\right)$$

Similarly, we compare team performances at a difference node d, which is distributed:

$$d \sim t_1 - t_2$$

The factor associated with this node connects to the t nodes and has the function

$$\mathbb{I}(d = t_1 - t_2)$$

A positive value of d represents a win for Team 1, while a negative value represents a win for Team 2. Note that performance difference here has no real-world interpretation. It is not tied to any actual performance metric.

This set of variables describes a generative model for performance differences. When we want to infer player skills, we would like to take evidence about the actual outcome of the game, represented by d, and propagate that information back up the factor graph. To do this, we use sum-product on the

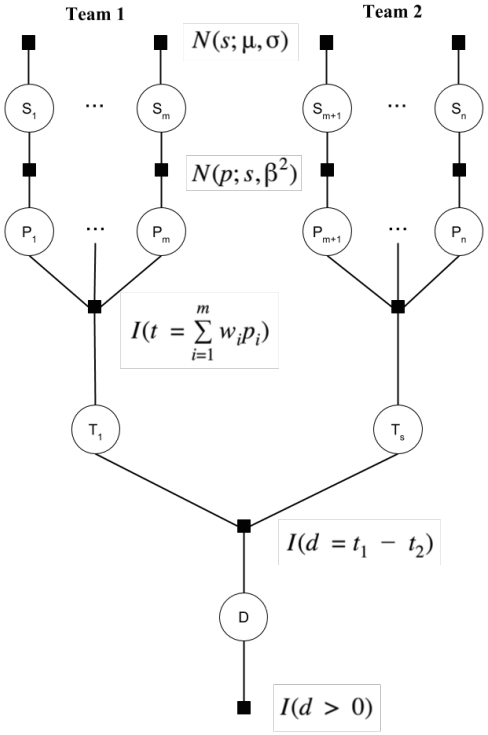


Figure 1: The TrueSkill factor graph for two teams. S nodes represent the latent skill for each player. A player's performance P is a Gaussian centered at his true skill, with known variance β^2. A team's performance T consists of the weighted sum of the players' individual performances. D compares the two teams' performances. A positive D corresponds to a win for Team 1.

factor graph to propagate information downward and get a prior over d. Since d is a difference of independent Gaussians, which themselves are sums of independent Gaussians, d is itself a Gaussian.

When we incorporate evidence into the graph (i.e. the result of the game), we truncate d so that its support equals the region that assigns positive probability to the true outcome. In other words, we assign zero probability to the side of the origin that represents the losing team. Propagating this message back up to the leaves of the graph amounts to taking products with a truncated Gaussian. Since this is difficult to do in closed form, we shift to approximate message passing and use expectation propagation. We approximate the posterior over d by a Gaussian method-of-moments estimator, and use this as the message passed up. We will describe message passing in more detail in the next section.

2.1 Sum-Product Messages

Message passing on the factor graph involves taking products of Gaussian distributed random variables. For simplicity, TrueSkill parametrizes these Gaussians in terms of the precision, $\pi = \frac{1}{\sigma^2}$, and the precision-mean, $\tau = \pi\mu$. If we have two independent random variables $x \sim \mathcal{N}(\tau_x, \pi_x)$ and $y \sim \mathcal{N}(\tau_y, \pi_y)$, then the random variable $z = x \times y$ has the distribution $z \sim \mathcal{N}(\tau_x + \tau_y, \pi_x + \pi_y)$ [5].

Skill inference on the factor graph consists of four steps. First, pass messages down to get a prior on D. Second, incorporate knowledge about the outcome of a game to get a posterior over D. Third, approximate this posterior with a new Gaussian via a method-of-moments estimator. Fourth, pass messages back up the graph using posterior values to update skill nodes S_i. The messages described here are taken directly from [1]. The equations assume all messages are initialized to 1, so they take on a simpler form than what is presented in the original paper.

The first message passed is from each prior node to its corresponding skill node. In Figure 1, the prior nodes are the boxes at the top of the image. The prior is parametrized in terms of μ_i and σ_i, so the message-passing update for s_i is:

$$\begin{aligned} \pi_{s_i} &= \frac{1}{\sigma_i^2} \\ \tau_{s_i} &= \frac{\mu_i}{\sigma_i^2} \end{aligned} \tag{1}$$

The performance nodes P_i are Gaussians centered at S_i with variance β^2, where β is a hyper-parameter controlling the spread in the distribution of realized performances based on skill. The prior over P_i is given by

$$\begin{aligned} \pi_{p_i} &= \frac{\pi_{s_i}}{1 + \beta^2 \pi_{s_i}} \\ \tau_{p_i} &= \frac{\tau_{s_i}}{1 + \beta^2 \pi_{s_i}} \end{aligned} \tag{2}$$

The team performance nodes T_i are weighted sums of the performance nodes of the players on the team. The weights, w_i, are parameters of the model. The message update for T_i relies on the mean and variance of the sum of Gaussians and on the definitions of π and τ:

$$\begin{aligned} \pi_{t_i} &= \left(\sum_{i=1}^{m} \frac{w_i^2}{\pi_{p_i}} \right)^{-1} \\ \tau_{t_i} &= \pi_{t_i} \cdot \left(\sum_{i=1}^{m} w_i \cdot \frac{\tau_{p_i}}{\pi_{p_i}}) \right) \end{aligned} \tag{3}$$

Finally, since D is the difference of T_1 and T_2, we can use the same technique as (3) to come up with our prior on D:

$$\begin{aligned} \pi_d &= \left(\frac{1}{\pi_{t_1}} + \frac{1}{\pi_{t_2}} \right)^{-1} \\ \tau_d &= \pi_d \cdot \left(\frac{\tau_{t_1}}{\pi_{t_1}} - \frac{\tau_{t_2}}{\pi_{t_2}} \right) \end{aligned} \tag{4}$$

The prior on D is a Gaussian that assigns probabilities to values of the difference in team performances, $T_1 - T_2$. Positive values of D correspond to a win for Team 1. To incorporate knowledge of the game outcome, we truncate this Gaussian to assign zero mass to values corresponding to wins for the losing team. In other words, if Team 1 wins, D is truncated and re-normalized to assign zero probability to values in $(-\infty, 0]$. If Team 2 wins, the same is done for $[0, \infty)$.

Passing this truncated Gaussian back up is difficult to do analytically, so TrueSkill uses expectation-propagation, which approximates the posterior over D with a Gaussian. This technique is part of a larger class of approximate message passing schemes that use moment matching to approximate messages [4].

The moment-matching approximation to the posterior over D [2] is given by a Gaussian parametrized by:

$$\begin{aligned} \hat{\pi}_d^{\text{new}} &= \frac{\pi_d}{1 - W(\tau_d / \sqrt{\pi_d})} \\ \hat{\tau}_d^{\text{new}} &= \frac{\tau_d + \sqrt{\pi_d} V(\tau_d / \sqrt{\pi_d})}{1 - W(\tau_d / \sqrt{\pi_d})} \end{aligned} \tag{5}$$

where the moment correction functions V and W are defined as:

$$\begin{aligned} V(t) &= \frac{\mathcal{N}(t)}{\Phi(t)} \\ W(t) &= V(t) \cdot (V(t) + t) \end{aligned} \tag{6}$$

Finally, the posterior over D is passed up the graph using the inverses of the downward messages to get a posterior belief over each player's skill. More details about the upward messages are available in [1].

2.2 Handling Match Sequences

The previous section outlined TrueSkill messages for a single match update. Now we will address updates over a sequence of games. The prior for each player's skill in a match is initialized to the posterior over skill from each player's previous match. However, for numerical stability a dynamics factor γ^2 is added to the prior variance of each player's skill. Omitting this term would eventually lead to variances collapsing to zero since the TrueSkill updates naturally tend towards smaller posterior skill variance as the system gets more confident in its belief of players' skills over time. Including the dynamics factor is also a sensible modeling choice since it accounts for out-of-match uncertainly, such as injury or fatigue, that may affect a player's skill between games. Mathematically, if $s_i^t \sim N(\mu_i^t, \sigma_i^t)$ is the posterior skill distribution for player i after applying updates from a match at time t, then that player's prior skill going into a match at time $t+1$ is $s_i^{t+1} \sim N(\mu_i^t, \sigma_i^t + \gamma^2)$. Note that for each player's first game, μ_i^0 and π_i^0 are initialized to μ_{init} and σ_{init}, which are model parameters.

3 TrueSkill RNN

Since TrueSkill updates often occur over sequences of matches, it is useful to reformulate the message passing procedure as the internal state of recurrent nueral network. Doing so yields a powerful abstract for overlaying deep components on top of the original TrueSkill machinery. Furthermore, redefining the message passing procedure as an RNN cell makes coding the algorithm simpler and more efficient when using autodiff libraries like tensorflow.

The internal state of a TrueSkill RNN consists of two vectors, μ and σ, parameterizing the skill Guassians for every player that appears in any of the games. At time t, the RNN receives as input two vectors denoting the indices of players on team 1 and team 2. These values index into the state vectors so that the proper skill priors are loaded into the factor graph. Downward messages are computed, resulting in a prior over D. This prior also serves as a prediction for the outcome of a match, so it is returned as the output of the cell at time t. Then upward messages are computed so that the internal state reflects the posterior over player skills after timestep t. This process continues for every match in a sequence. Figure 2 visualizes the RNN as well as the internals of a TrueSkill cell.

3.1 Training and Backpropogation

By reformulating TrueSkill as an RNN, we can run backpropogation through time to learn suitable values for model parameters like β and γ. While message passing defines the training procedure for learning beliefs over player skill, backprogation defines a training procedure for learning model parameters. This is possible since the message passing dynamics are fully known and deterministic, meaning we can differentiate messages with respect to each parameter. Motivated by our interest in predicting game outcomes, we can construct a training loss using maximum likelihood:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} [y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})] \tag{7}$$

where $\mathbf{y} \in \{0, 1\}^n$ is a vector of indicators denoting whether team 1 won match i and $\hat{\mathbf{y}} \in [0, 1]^n$ is a vector of predicted probabilities of team 1 winning match i. Additional details on training are provided in section 6.

4 High-Dimensional TrueSkill RNN

TrueSkill RNN is the same model in form as vanilla TrueSkill besides the added ability to train model parameters with backpropogation. In order to extend the framework for high-dimensional games, we need a method for incorporating features specific to each player's matches. A natural place to do so is to use the weight parameters from (3). The team performance node is a weighted sum of

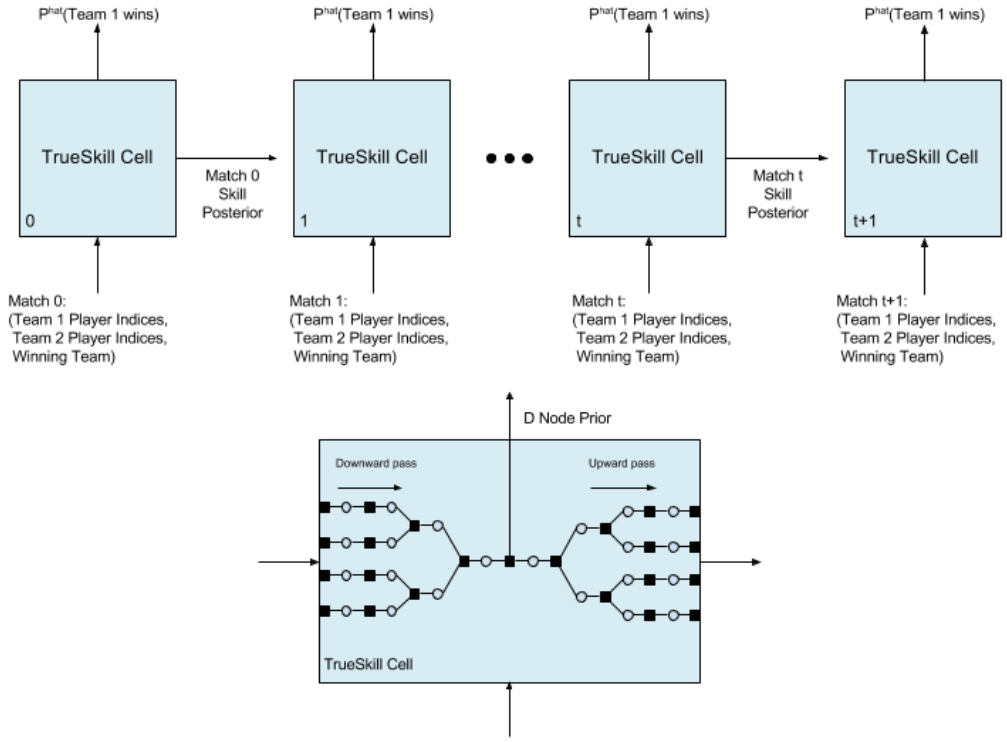


Figure 2: The vanilla TrueSkill algorithm reformulated as a recurrent nueral network. Each RNN cell performs the downward and upward message passes for a single match update. The internal state of the RNN is (μ, σ), both vectors parameterizing player skill distributions with length equal to the number of players that appear in any match. At each timestep, the cell outputs a single probability, taken from the D node prior, denoting the prediction for a match at that timestep.

players' performances. Weighting each player's contribution to a team is a sensible decision since not every player on a roster influences a team's overall outcome equally. In basketball, the five starters are generally more influential to a match's outcome than players substituting in from the bench. If a boxscore consisted of a single statistic for each player, weighting players would be straightforward: simply use the statistic, or perhaps its normalized value, for the weight. However, since by definition high-dimensional games have boxscores with many statistics for each player, such a system will not work. Instead, we have to learn a function, $\mathbb{R}^p \to \mathbb{R}_+$, to map boxscores with p features to weights. Specifically, we propose using a two hidden layer, fully connected, feed-forward neural network with ReLU activations, as the function approximator. Each hidden layer has 64 nodes and the final output layer is also passed through a ReLU activation to ensure that the TrueSkill weights are non-negative.

The High-Dimensional TrueSkill RNN (HDTSRNN) model extends TSRNN by incorporating boxscore features when calculating skill posteriors following a match. At each timestep, in addition to the team 1 and team 2 player indices and winning team indicator inputs, the RNN cell receives two tensors with player boxscores from both teams. Each tensor is normalized along feature axes, so that a player's weight is computed based on his or her relative contributions. For example, rather than considering the total points scored by each player, normalizing the boxscore tensor results in the network considering the fraction of team points scored by each player. Normalized values are better to work with since the weight network is trying to learn a player's relative contribution to the team with respect to each facet of the game.

Rather than learning a single function to map between player boxscores and weights, we instead learn two: one for the winning team and another for the losing team. Learning two functions allows for the model to build an understanding of how credit among team players should be assigned depending whether the team ultimately wins or loses. Committing many fouls, for example, may be a good or bad thing for a player depending on how his or her team fares.

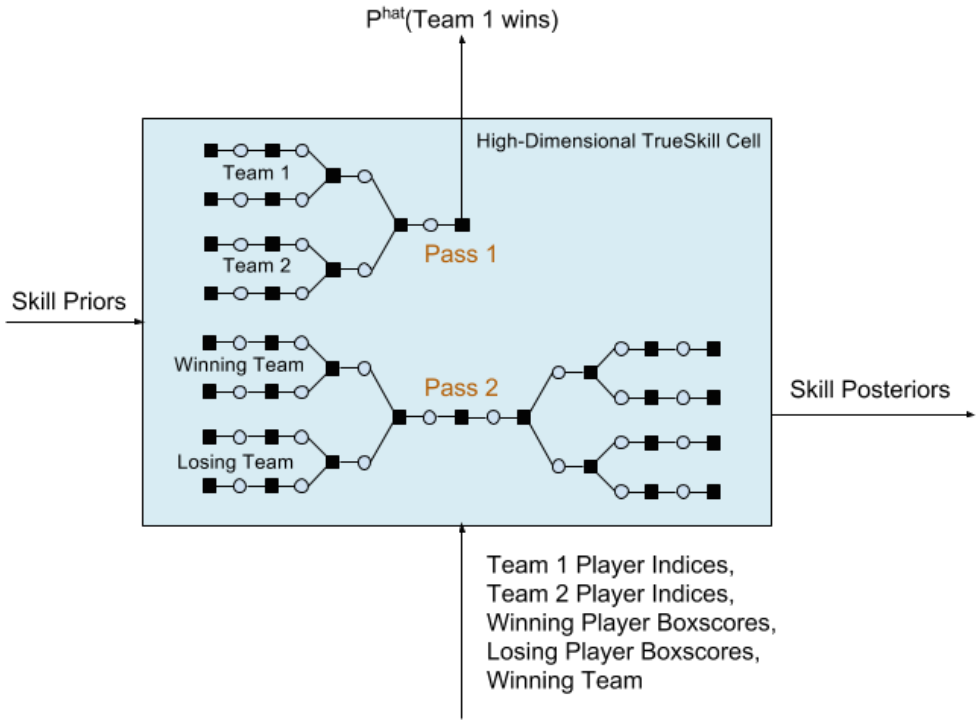


Figure 3: The High-Dimensional TrueSkill RNN cell. First a downward pass (1) is conducted with uniform TrueSkill weights to produce a prediction for the game. Then, a full downward and upward pass (2) is conducted to produce skill posteriors following the game. Pass 2 feeds winning and losing player boxscores through their respective neural networks to map boxscores to TrueSkill weights.

The final, and perhaps most subtle, detail in defining the HDTSRNN model is necessary for avoiding look-head bias: weights used at inference-time cannot consider boxscore features from the very game the model is trying to predict. Thus, for a cell at time t we can only use the boxscores from game t to perform skill updates and not to perform prediction. Therefore, inside each HDTSRNN cell we first conduct a downward pass using uniform weights to arrive at a prior over D, which becomes our prediction, and then conduct a full downward and upward pass using boxscore data to determine update weights for computing skill posteriors. Because the training loss is a function of predicted values, training HDTSRNN will teach the underlying TrueSkill algorithm how to update skills in order to improve likelihoods of future games. The model learns how to best take boxscores from a game at time t to determine optimal skill updates in order to improve predictions for games at times $t + 1$ and beyond. Figure 3 visualizes the HDTSRNN cell.

5 Dataset

We scraped player boxscore and game outcome data from all NBA games occurring during the 12 seasons ending in years 2006 through 2017 from Basketball-Reference.com. Our data consist of boxscore records for 1,273 players from 15,524 games. Table 1 lists and describes the 15 player boxscore statistics we consider.

Feature	Description
sp	seconds played
fg	field goals made
fga	field goal attempts
fg3	3 pointers made
fg3a	3 pointer attempts
ft	free-throws made
fta	free-throw attempts
orb	offensive rebounds
drb	defensive rebounds
ast	assists
stl	steals
blk	blocks
tov	turnovers
pf	personal fouls
plus-minus	net score differential while playing

Table 1: Player Boxscore Features

6 Approach

To evaluate our model, we compare HDTSRNN to vanilla TrueSkill as well as to a few baselines. This section will address evaluation metrics, baseline methods, NBA specific adjustments to the underlying TrueSkill framework, and further specifics about training HDTSRNN.

6.1 Evaluation Metrics

A simple approach for evaluating match prediction systems would look at the fraction of matches a model "got right." A correct prediction is a game where the team that actually won received at least a 50% predicted chance of winning. This metric is flawed, however, since it does not consider a model's confidence in its predictions. Consider two games where Team 1 wins the first and loses the second. Model A predicts Team 1 will win both games with 3% and 45% chances, respectively. Model B predicts Team 1 will win with 35% and 45%, respectively. Both models were 50% accurate. They missed the first game and correctly predicted the second. Nevertheless, A was much more confident in its incorrect decision for game 1 than B. Ideally, an evaluation metric would consider this observation. Therefore, we propose evaluating based on likelihood: the model's predicted chance of observing what actually happened in the real world. Likelihood methods reflect a model's confidence, and therefore can further distinguish two methods beyond accuracy. In this paper we report log-likelihoods since they are numerically easier to calculate. We also report accuracy since log-likelihoods are hard to intuitively interpret while accuracy can still paint a crude picture of how a model fares.

6.2 Baseline Methods

We propose two naive baseline methods to ensure that vanilla TrueSkill and HDTSRNN outperform techniques that are much simpler to understand and easier to implement. The first is *home-team-predictor*, which predicts that the home team will win with probability equal to the empirical home-team win rate from the period it is trained on. The second is *overdog-predictor*, which predicts that the team with the better record will win with probability equal that team's season win rate prior to the game in question.

6.3 NBA Specific Adjustments

For both the vanilla TrueSkill and HDTSRNN, we adapt the underlying TrueSkill framework to better reflect NBA conditions in two ways. First, between each season there are several factors that lead to new uncertainty over a player's skill. Trades, aging, and rule changes, among others, give reason to be less confident about a player's skill at the beginning of a season as compared to the models belief at the end of the previous season. To reflect these dynamics we introduce λ^2, a model parameter

Method	Test Accuracy	Test Log-Likelihood
Overdog Predictor	61.96%	-916.98
Home-Team Predictor	58.29%	-890.03
Vanilla TrueSkill	63.41%	-836.47
High-Dimensional TrueSkill RNN	**64.63%**	**-826.96**

Table 2: Model Results on the 2017 Test Season

added to every player's skill variance at the beginning of each season. Mathematically, λ plays a similar role to γ in that both terms add uncertainly to a player's skill through time. Second, because home-team bias is a real phenomenon that is not captured by TrueSkill we add a linear term, η, to each predicted output from the RNN if team 1 is playing at home and subtract eta if team 1 is playing away. Both λ and η are learned using backpropogation like other model parameters.

6.4 Training Details

Since we are primarily interested in training the weight networks to learn how to best conduct TrueSkill updates, we need to ensure that gradients do not vanish as they are being propagated back through a long RNN. To combat this problem, we train in batches of 250 consecutive games. The internal state of the RNN is carried over between batches, however at the beginning of each epoch the state is reinitialized so that every player's skill parameters are set to $\mu_{init} = 25$ and $\sigma_{init} = \frac{25}{3}$, default values taken from [1]. Reinitializing between epochs is important so that skill values learned from later games are not then used to predict earlier games. Time-series models are difficult to train because decisions made at time t affect all matches from $t + 1$ onward, and therefore affect future decisions. For this reason, we cannot train on random batches of games. Furthermore, because the underlying TrueSkill model is deterministic, there is no source of randomness in the training procedure. To avoid converging at poor local minima, we must on many random seeds to initialize neural network weights and TrueSkill model parameters. We use a validation approach to pick final model parameters because cross-validation is difficult on time-series data for the same reasons as using random batches.

7 Results

Models were trained on data from the 2006 to 2015 seasons, validated with the 2016 season and tested on 2017. Test results are summarized in Table 2. The HDTSRNN model outperforms vanilla TrueSkill along with the two baselines both in terms of accuracy and log-likelihood. There were 1309 games played during the 2017 test interval. In addition to being more likely, HDTSRNN correctly picked 16 more games than the next best approach.

8 Conclusion

We have shown that the TrueSkill algorithm can be improved for high-dimensional games by incorporating extra information specific to players from a match. Such boxscore features are useful for learning a model to assist TrueSkill in weighting updates following a game. By reformulating TrueSkill as recurrent neural network, we are able overlay deep learning techniques upon an underlying graphical model, leveraging a neural network's powerful expressive capacity with the interpretability of a factor graph. The method we proposed is general for any multiplayer competitive activity and could be used build prediction systems for other professional sports leagues.

Because TrueSkill is able to evaluate players individually within team competitions, our framework can eventually be used to build systems for determining relative player value for a franchise during the course of a season. Such a tool could inform roster construction and trades within a professional league. We leave this kind of analysis to future work.

References

[1] Ralf Herbrich and Minka, Tom and Thore Graepel. (2007) TrueSkill™: A Bayesian Skill Rating System. In P. B. Schölkopf and J. C. Platt and T. Hoffman (eds.) *Advances in Neural Information Processing Systems 19*, pp. 569–576: Cambridge, MA: MIT Press.

[2] Ralf Herbrich. (2005) On Gaussian Expectation Propagation. https://www.microsoft.com/en-us/research/publication/on-gaussian-expectation-propagation/.

[3] F. R. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. IEEE Trans. Inform. Theory, 47(2): 498–519, 2001.

[4] Wainwright, Martin J. and Jordan, Michael I. (2008) Graphical Models, Exponential Families, and Variational Inference. pp. 1–305: Hanover, MA, USA: Now Publishers Inc.

[5] Bromiley, P. A. (2003) Products and convolutions of Gaussian probability density functions. Tina-Vision Memo 3.