Generating NFL Game Headlines from Box Score Statistics

Parker Greene, **Itamar Belson**, and **John Clarke**
{pwgreene, ibelson, jaclarke}@mit.edu

Abstract *In this paper, we introduce a new method by which to extrapolate key game statistics from a complete, purely numerical box score in order to produce a one-sentence summary of the game. Through identifying the particular game figures of greatest importance, the system is able to generate informative headlines that effectively highlight the most significant elements of the particular game. We describe a system that: (a) successfully utilizes a neural network to identify the significant elements of information necessary to summarize the complete set of data; (b) uses the important data components to generate complete, comprehensive headlines. In this paper we explore the application of the methods to football games played in the National Football League (NFL), but the same approach may be applied to other sports or to a number of other applications.*

I. Introduction

Sports analytics is an emerging field that seeks to incorporate principles of big data analytics into the world of sports. Data in this arena is extensive, and largely publicly accessible, due in part to the economic appeal of the sports industry as well as the popular public interest in sports. Yet, although considerable work has been completed in the field, much of the work centers on game prediction and individual player evaluation as these applications provide tangible economic benefits to the team franchises. While work for these applications is ongoing, there has been significantly less work in the process of analyzing a complete set of statistics to extrapolate the significant components within a particular game.

Whether a key player's performance or the overall struggle in the contest, concluding such information currently requires either watching the complete game or a comprehensive understanding of and ability to read thorough a publicly available box score. A popular alternative to the two is to read game summaries written by sports professionals, whose expertise in the matter enables them to recap a complete game in a condensed form that is understandable to any reader.

This paper explores the potential of machine learning and natural language processing approaches and techniques to extract the important game performances from the box score of NFL games in order to automatically generate these summarizing game headlines. In particular, the paper explores various approaches consisting of neural networks and a variety of generative models to develop an effective system that is capable of completing the currently manual task. Some challenges faced by the system include summarizing a game from a largely naive data set that, in actuality, gives but a glimpse into the complete time-dependent game riddled with the intricacies common to sports. To help combat such issues, we simplified the problem statement to only involve the summarization of events in the actual game, without incorporating information that would require citation outside of the box score. As a result, injury reports, playoff contention, and game disputes are not incorporated into the headlines generated by the developed system.

A. Motivation

From sports to Wall Street, extrapolating key figures from a restricted set of data is an ongoing area of research with applications in a variety of fields. In addition, summarizing the identified key figures in a comprehensive headline may prove instrumental to fields in which language proves more powerful than numbers. As such, although this paper focuses on the information extraction and summarization of football games from box scores, the methods described in this paper may by applied to far reaching applications.

II. Related Work

Generation of somewhat short sentences using Markov chains and stochastic methods have been shown to be quite simple and effective in generating poetry [1] [2], so we focused on applying these same principles in synthesizing valid headlines. This method yields a good way for text generation learned from examples; however, there is relatively much less research in the topic of generation solely from a combination of statistics and their corresponding text examples. Other, more syntactically-based models geared toward dialogue systems [3] [4], we found to be inapplicable to our problem due to the lack of grammatical structure of the headlines in our data. Wen et. al. [5] presented an empirically-tested LSTM model capable of generating linguistically varied responses, but applying a deep, recurrent model to our problem proved hard due to the lack of data and difficulty of integration of the statistics in training.

Football game summaries have been explored by Nichols et. al. [6] and Chakrabarti & Punera [7], but both methods relied on learning data directly from Twitter and were uninformed by the statistics of games, as our model was.

III. Data Collection

The data used by the system was extracted directly from ESPN's publicly accessible website1. The website contains comprehensive historical statistics for the past ten seasons of NFL games (2007-2016), with a complete box score for each game played throughout each season. From these box score, we extracted twenty-eight key statistics that would provide substantial insight into each overall game. In addition to the statistics extracted directly from the box score, we also incorporated team specific information for each game, such as the teams' city and acronym. This information proved useful in the annotation of the game headlines described below.

In addition to the box score data, each game on the website also has an accompanying article headline that was manually written by a professional sports writer following the game. We scraped this headline for each of the games for use as training data for our system. However, as described in the next section, our approach took into consideration not the headline provided directly from the website, but a general form of the headline. To generalize the headlines, we removed all game specific names and figures and replaced them with identifying tags. A few examples of the general form headlines can be seen in listing 1. The initial process was completed automatically but required additional manual processing to ensure accurate annotation of the data. After the data was compiled and annotated, we removed any game data with an accompanying headline that incorporated information external to the actual gameplay, such as injury reports, playoff contention, and game disputes. In conclusion, our final dataset consisted of data and a headline from 1742 individual games.

```
"Kolb throws 2 TDs to lead Eagles' rout of Chiefs.
"[team_1_leader_passing] throws [team_1_leader_pas
```

Listing 1: An example of a training headline (top) and its corresponding annotation (bottom)

IV. Approach

Our implementation is made up of two separate models, a neural net (Section IV.B) to map the size statistic vector s into a word-likelihood vector d and a text generator which we refer to as the Re-weighted Trigram Markov Generator, or RTMG (Section IV.C). An overview of the entire system is shown in figure 1.

A. Statistic Feature Vectors

We explored a variety of options for encoding the raw statistics into feature vectors. The first, a naive approach, was just the 18 numerical statistics from the box score data, and we found, as exprected, that it under-performed more complex representations. The second option was a concatenation of 1-hot encodings of the statistics to test a higher-dimensional separation of the data. This representation under-performed our optimal feature representation due to the sparsity of the feature vectors. We sampled various combinations of statistics using our priors on the structure of the data. After testing the representations at their optimal learning rate, we identified the features illustrated below. Some values were chosen unmodified, the others converted into a 1-hot classification vector to determine the numerical range in which the statistic lies. For example, we bucketed passing yards into

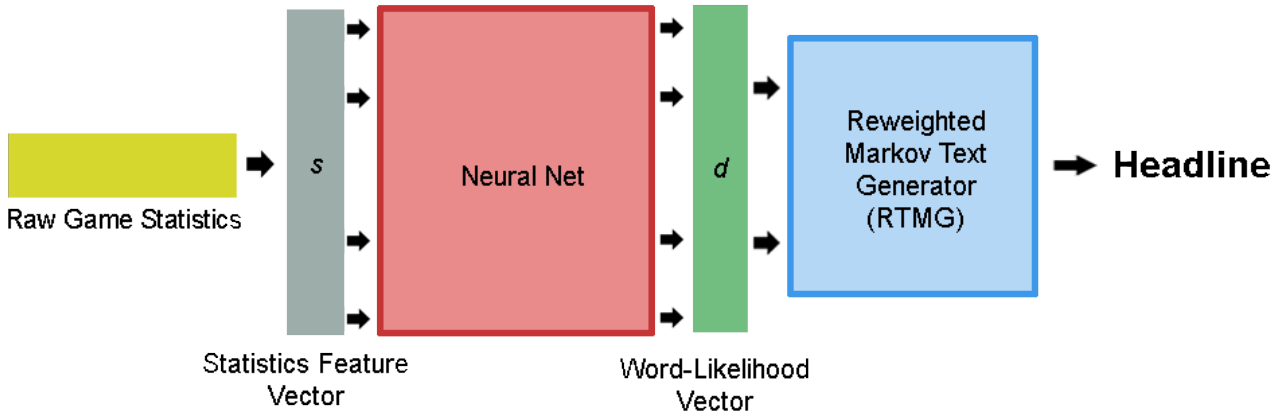

Fig. 1. Box Diagram of the full headline generation system

50 point ranges from $0-500+$ representing ranges $[0-50, 50-100, ..., 500+]$.

- team_1_leader_passing_td value
- team_1_leader_rushing_td value
- team_1_leader_receiving_td value
- team_1_leader_passing_yds $[0-50, 50-100, ..., 500+]$
- team_1_leader_rushing_yds $[0-25, 25-50, ..., 150+]$
- team_1_leader_receiving_yds $[0-25, 25-50, ..., 150+]$
- game_leader_kicker_points $[0-1, 1-2, ..., 3+]$
- game_leader_scorer_points $[0-1, 1-2, ..., 3+]$
- team_1_leader_passing_int value
- team_score_diff $[0-2, 2-7, ..., 21+]$

We found that statistics with magnitudes $>= 50$ should be cast as range classification vectors to decrease the dissimilarity of close numbers. Similarly, we did not cast low values < 50 as classification vectors but rather took their values. The majority of statistics focused on team 1, the winning team, as we found headlines generally were biased to the winning team. Furthermore, the optimal representation is comprehensive, covering offense, defense, and special teams of NFL football teams.

B. Neural Net

The NN accepts a vector, s, of statistics and returns a vector containing the probabilities of words existing in the headline associated with those statistics. First, the NN projects the statistics vector s through two hidden layers of size $2|s|$. The hidden layer outputs are activated by a Rectified Linear function (ReLU). Then, the result is projected to a vector space of size $|V|$, where V is the vocabulary including start and end characters. Finally, the output is activated by a softmax function. Refer to 2 for an illustration of the NN architecture. We initialize the weights by sampling from a Gaussian distribution $N(0, 1)$. To train the model, we optimize a categorical cross-entropy loss function using Adam. This training method suits our objective of producing multi-class predictions. Lastly, we regularize our model on two fronts using: (1) Adam, a self-regularizing gradient descent method, and neural dropout, where we ignore the weights connected to neurons that are "turned off" with probability p during training. With our vector y of word probabilities $p(w_i)$ for $w_i \in V$, we move on to trigram MC.

C. Re-weighted Trigram Markov Generator

We initialize the RTMG with transition probabilities estimated from the corpus of 1742 game headlines. For this task, we simply calculated the maximum likelihood estimates for each word as $p(w_i|w_{i-1}, w_{i-2}) = \frac{count(w_i, w_{i-1}, w_{i-2})}{count(w_{i-1}, w_{i-2})}$, yielding a matrix of transition probabilities, T. Refer to Figure 2 for an example illustration of the MC model. At this point, the MC could be used to create reasonably semantically and syntactically correct sentences. The model takes an element-wise prod-

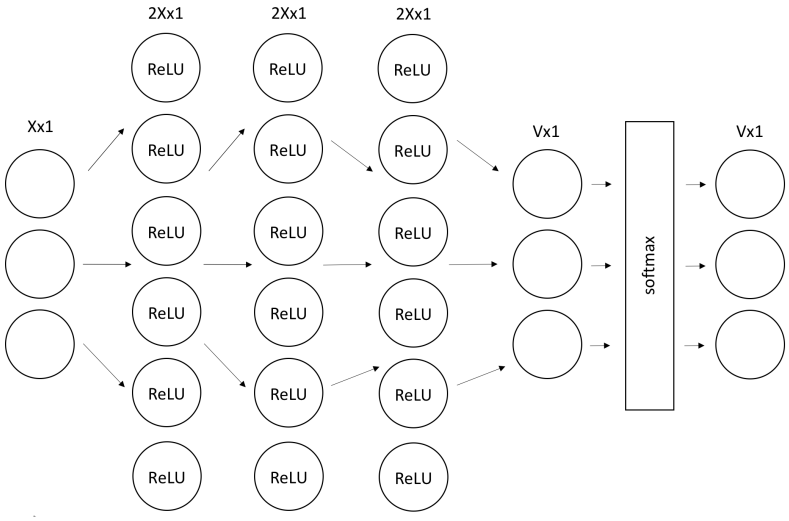

Fig. 2. Overview of the neural network architecture. Input to the network is the feature vector s and output is the word-likelihood vector d.

uct of our word probability vector d from (1) and each row of T, which describe the transitions $u, v \rightarrow w$, for all words $w \in V$ given words u, v. These probabilities serve as discounting factors to properly weigh down the transition probabilities to words unlikely to appear in the headline. Leveraging this information results in generally grammatical sentences that are generated around the relevant vocabulary. Code for the re-weighting algorithm in showing in listing 2. The function takes as input T, the existing transition matrix as estimated from the corpus of headlines. T is re-weighted by d, which is the output of the neural net and has the same length as each row of T. The re-weighted transition matrix, M is returned. Note that d changes based on s, the input feature vector to the neural net, so T is re-weighted only to generate sentences for a particular s and then re-weighted differently for a different s.

Sentences are generated by doing a random walk over the possible states, where each word, including the start and end symbols, are considered states. The generator starts at a pair of two start symbols and then randomly walks over the words according to the transition probabilities until the end symbol is reached. If the current state is (u, v), then the next state is chosen according to the probabilities in M corresponding to pair (u, v).

```
def reweight_markov(d, T):
  M = copy(T)
  for u in range(0, n*n):
    for v in range(0, n):
      M[u][v] = T[u][v] * d[v]
 return M
```

Listing 2: Re-weighting the Markov text generator

V. Results

We consider the neural network and the overall system (NN+MC) in our results and analysis. This is because it is important to show that the NN conveys to the MC the likelihood of words to appear in the headline. We partitioned the statistic and headline data into 70% training (1,219) and 30% test data (523). We trained the neural network until convergence, using the optimal batch size and number of epochs. For different numbers of hidden layers and neural dropout probability, we calculate the cross entropy-loss. More over, for the T test examples, we calculate a quantity similarity = $\sum_{t=1}^{T}$ Intersect(Top-K-Words($S^{(i)}$),$H^{(i)}$) / $\sum_{t=1}^{T} H^{(i)}$.length, where Intersect calculates the intersection of Top-K-Words–the top k words by likelihood predicted by the neural network from the statistics features $S^{(i)}$ and the top k words in the test headline, $H^{(i)}$. This metric measures the similarity of the k words in the test headline with the most likely k words according to our neural model. The loss and similarity results are reported in Table II.

The cross-entropy loss reflects the similarity of the predicted word-likelihood vectors and the input probabilities. As the number of hidden layers in-

TABLE I
THE NEURAL NETWORKS WERE TRAINED WITH AN INITIAL LEARNING RATE OF 0.05 USING ADAM.

Num. of Hidden Layers	Dropout Probability	Loss	Similarity
1	0.10	3.76	0.42
2	0.10	3.62	0.45
3	0.10	3.47	0.49
1	0.25	3.69	0.44
2	0.25	3.61	0.45
3	0.25	3.41	0.54
1	0.50	3.87	0.35
2	0.50	3.67	0.41
3	0.50	3.52	0.45

creases, the loss decreases across all dropout probabilities we tested. Despite the increasing complexity, our model does not degrade due to the increasing difficulty of optimization. The most performant model used three hidden layers and a dropout probability $p = 0.25$. The last batch of models experienced too high of regularization, with the the models failing to learn from the data optimally.

Our system sought to map a feature vector of numerical statistics to a headline describing the game. We evaluated performance of the whole system by measuring the similarity of the headlines predicted on the test statistics and the T test headlines. This was found by first generating 30 sentences per test statistic vector with the MC. Then, we calculated the maximum similarity = $\max_{i=1...30}$ Intersect($y^{(i)}(S^{(j)})$, $H^{(j)}$) / $H^{(j)}$.length for i = 1...30, where $y^{(i)}(S^{(j)})$ represents the ith prediction on a fixed statistic vector and Intersect measures the intersection this and the associated test headline. The maximum similarities for all test data were then averaged. We take 30 measurements to reduce the inherent variance in the MC, which relies on a sampling of a discrete trigram transition distribution to determine the headline. We rejected all prediction headlines with a word count greater or less than five words from the test headline's count. It is important to note that this metric does not capture the MC predicted headline's similarity to other training headlines from which it learns its initial transition probability matrix. In fact, many of our longer, more descriptive predicted sentences appear to borrow from a number of training examples, despite being biased toward the words predicted by the neural networks. Results for the bigram and trigram markov models are reported in Table 2.

TABLE II
WE USE THE OPTIMAL NEURAL MODEL ILLUSTRATED IN FIGURE 2 TO PREDICT WORD LIKELIHOOD VECTORS. WE THEN RE-WEIGHT (RW) THE TRANSITION MATRICES OF THE BIGRAM AND TRIGRAM MODELS.

Model Type	Avg. Headline Word Length	Similarity
Bigram MC	3.37	0.197
Trigram MC	5.21	0.187
RW Bigram MC	3.11	0.243
RW Trigram MC	4.97	0.232

As expected, Trigram models tend to produce more complex sentences with lengths greater by nearly a factor of 2. Furthermore, the similarity of re-weighted models is greater. This is because by re-weighting the initial transition matrix, we bias the predicted headlines toward words included in the test headline we compare against, given that our neural network predicts these words with reasonable accuracy.

VI. CONTRIBUTION

A. Parker Greene

I started by working on the model design, and after a lot of research into text generation, I initially came up with the idea of not using the raw statistics as input directly to the sentence generation model, but instead separating the system into the two separate models: one for creating an embedding from the statistics and one for using this embedding to generate the headlines. After the initial model design, I focused on the problem of text generation. The initial approach I considered was the bigram Markov chain, which had the ability to generate sentences, but could only learn directly from the corpus of headlines and thus relied on the neural net output. I also explored the possibility of using both word-level and text-level sequence generation using an LSTM network. This approach worked well for learning from the corpus how to generate sentences, but unlike the Markov model, it could not easily be integrated with the existing embeddings output by the neural net. I also explored the possibility of using PCFG's to generate the headlines, which would have been able to assure the headlines had a given structure, but, as was touched upon in section III, I found they could not properly work with the headlines we had, since the headlines did not follow

proper grammar. We contributed equal work to this write-up and the presentation.

B. Itamar Belson

I started by researching existing work and models in the field of text generation as it pertained to our proposed problem. After compiling various ideas, we met as a group to discuss the benefits and the drawbacks of each approach. My main contribution to the project was the collection and annotation of the data. I developed a web scraper that would collect the historical data from ESPN?s website. This presented several challenges, as the data was not consistently formatted between seasons. After collecting all of the necessary data, my next task was to annotate the data by properly tagging each of the headlines and removing all referenced names and numbers. This process was done as a two-step process that consisted of automatic information extraction tagging and then manual assessment of each of the over two thousand headlines to ensure proper annotation of each data element. As a final step, I reviewed each of the headlines to identify any headline that referenced events external to the game, a key constraint of our system. After collecting and annotating the data, I helped the rest of the group in various ways. First, I worked with John to develop an ideal feature vector to optimally encapsulate the game statistics. This consisted of several iterations to ensure an optimized feature representation for the neural network. I also worked with the entire team to brainstorm system improvements and model specifications and helped solve several bugs along the way. Finally, I contributed equally to all team members in the final write-up and presentation.

C. John Clarke

I began by researching the design of neural networks used for multi-classification tasks. I iterated through numerous different neural network architectures (number of hidden layers, number of neurons, activation functions), evaluating them on the training data. I eventually implemented the optimal neural network used to transform statistic vectors into word likelihood vectors. I collaborated with Itamar on identifying the important features for our vector representation of the data. I worked with Parker to connect the neural network outputs to his markov models to generate headlines. After, we developed basic statistics to evaluate the success of our generative model where a specific metric was not obvious. We all collaborated on the write-up and presentation.

VII. Source Code

Code is available at https://github.com/pwgreene/6.864-Final-Project

References

[1] Kenner, Hugh; O'Rourke, Joseph (November 1984). "A Travesty Generator for Micros". BYTE. 9 (12): 129-131, 449-469.

[2] Hartman, Charles (1996). *Virtual Muse: Experiments in Computer Poetry*. Hanover, NH: Wesleyan University Press.

[3] Alice H. Oh and Alexander I. Rudnicky. 2000. "Stochastic language generation for spoken dialogue systems". In Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3, ANLP/NAACL-ConvSyst '00.

[4] Adwait Ratnaparkhi. 2002. "Trainable approaches to surface natural language generation and their application to conversational dialog systems". Computer Speech and Language.

[5] Tsung-Hsien Wen, Milica Gasic, Dongho Kim, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015. "Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking". In Proceedings of SIG-dial. Association for Computational Linguistics.

[6] Jeffrey Nichols, Jalal Mahmud, Clemens Drews. 2012. "Summarizing Sporting Events Using Twitter". IBM Research - Almaden

[7] Chakrabarti, D., and Punera, K. 2011. "Event summarization using tweets". Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media.