Deep Learning: Word Embeddings for NBA News

Pau Bramon

June 14, 2018

Contents

1 Introduction

The aim of this report is studying and understanding how word embeddings work and how much information one can extract from them when used in a very specific context. Therefore, in this report we will try to extract information from NBA basketball news, with no other information than raw text. The idea is studying if the embedding, without prior information, is able to relate and group together concepts like players, teams, actions of the game, etc.

In the first section, a brief description of the dataset is given. Secondly, the main aspects of the implementation of the embedding network will be described; how the raw data is pre-processed and the main parts of the network will be analysed. Finally, some results and properties of the obtained embedding will be shown, and some possible applications of it will be studied. At the end, some conclusions of the work done and possible future improvements will be given.

2 Dataset

For the experiments proposed in this study, the corpus chosen to train the embedding is related to NBA basketball news. The data was obtained scrapping the webpage *http://www.rotoworld.com* [1], containing a compilation of tweets and short news in its section *Player News*. These news contain a short text, summarizing the news using up to 360 word, and a long text describing what happened in more detail using up to 956 characters. While the short texts are shorter and cleaner (syntactically correct and without typos), some of the long texts contains much more incorrectly written words, abbreviates and acronyms, literal citations with slang text, etc.
A total of 26160 news were collected from season 2017-2018, from September 12th to June 11th (starting slightly before the start of the season and ending some day after the end of it). These text were cleaned and preprocessed as it is explained in Section 3.1.

3 Implementation

In this section, the implementation of both the preprocessing part and the embedding part will be described. The whole implementation has been done in Python 2.7 using TensorFlow for the network implementation and it is available in *https://github.com/paubramon/nba_embedding*.

3.1 Data preprocess

The first step to use the data collected from [1] is sorting and merging all valuable data. First of all, the short and long texts will be filtered in order to replace or eliminate the following symbols:

1. Non-English characters: characters like greek letters and foreign symbols like ö/ő/ê/ë among others, will be replaced by its homonym in English.

2. Punctuation Marks: almost all of them will be removed, except for some like the dash or the apostrophe. The apostrophe is a common way to create a negative or possessive sentences and might be interesting to see if the embedding finds similarities between those forms. The dash is a common way to express percentages in basketball (for example one may have hit 5-10 shots from the field) and we will see if the embedding is able to group these forms together.

After filtering the two sets of texts, we will merge them to have a single dataset for the experiments. Therefore, our dataset will be a set of 52320 texts of different lengths. Note that multiple experiments were performed using both or only one of the two sets (the short and cleaner texts or the long ones), but we couldn't find a way to quantitatively measure which one was better. The experiments in Section 4 were done using the two sets, since it qualitatively seemed to give better results, probably because the dataset was larger.
Furthermore, note that in training, we have to keep the texts as independent instances, since mixing them we would be mixing totally different contexts. In the following section, when describing the implementation of the embedding, we will address this problem when creating the batches to train the embedding.

Finally, we will do a subsampling of some of the most frequent words. In large corpora, there are some words that show up very often but don't

provide much context to the nearby words, (e.g. "the", "a", "of", etc). As it is stated in [2], the usual words provide less information than the rare words, since we will find co-occurrences of these with other words all the time, since they are frequently present in most of the sentences. In embeddings, we want to give more importance to co-occurrences of less common words, like for example the name of a player with the name of the team where he plays.
In order to counter this imbalance between common words and rare ones, [2] proposes to filter all words with a probability computed by the following formula:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{1}$$

where $P(w_i)$ is the probability of not discarding the word i, $f(w_i)$ is the frequency of word i and t is an arbitrarily chosen threshold. For the following sections, we will use a threshold such that the most common word ('the' for our corpus) will have a probability of 0.5 of being discarded.

3.2 Embedding Implementation

The embedding studied was the Skip-Gram word2vec model by [2]. The implementation was done in TensorFlow, following a similar approach to the one used in [3]. The implementation will be described in two parts: network description and batch generation.
The first part briefly describes in general terms the architecture and the optimization used to obtain the embedding. The second part focuses on the batch generation step and the different variations used.

3.2.1 Network Description

The model used to learn the word embeddings from raw data is the Skip-Gram word2vec model, which tries to predict context words from a given target word. In order to do so, we will create a network with a single hidden layer, where the input will be the target word (using the embedding_lookup function from TensorFlow to do the equivalent of the one hot encoding) and the output will be the prediction of the context words.
Since we are only interested in the first part of the network (the hidden activations for each word), we will not compute the loss with all possible words in the vocabulary at each iteration, since it would be computationally very expensive. Instead, we will use the so called Noise-Contrastive Estimation

(NCE) loss, which measures how well the network is able to discriminate real context words from some randomly picked negatives samples (words that are not part of the context). So for each iteration, we will present a target word to the network and we will check how well the network is able to discriminate context words from others.
In order to train the network we will minimize the NCE loss with the Adam descent method,

3.2.2 Batch Generation

At each iteration we need to generate a batch of pairs of <target,context words> to train the network. In order to do so, for each word in the texts, we want to grab all words in a windows around that word. So for each learning step, we will pick a batch of N instances (for our experiments we used 128) containing a target word and a number of words within a windows around the the target. The window will be usually bigger than the final number of words used for training, where we will randomly select a subsample of them for training. Figure 1 shows an example of one element of the batch, where the center of the window is the target word and the context words for prediction are randomly selected from within the windows. The next instance of the batch will be the same figure shifted one word.

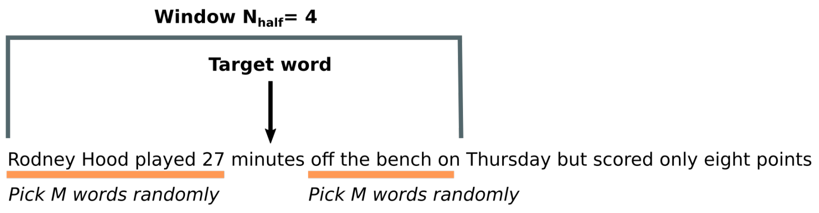
Window N_{half}= 4
Target word
Rodney Hood played 27 minutes off the bench on Thursday but scored only eight points
Pick M words randomly
Pick M words randomly

Figure 1: Example of batch generation with window size four words at each size of the target.

Note that multiple combinations can be used as batches. We have to choose a window size and the number of randomly picked words for training. Intuitively, having a larger window size will allow us to search longer dependencies, but it will probably decrease the relation between embeddings of consecutive words, since they will be presented together less often to the network. These are parameters we could change depending on the problem we needed to solve with the embedding.

4 Results

In this section we will train an embedding to see if it is able to find interesting relations between words. For this experiment, we will train a network with an embedding dimension of 200 (so we will represent each word with a vector of 200 elements), a batch size of 128 instances, 64 negative examples to compute NCE loss and a window size of $N_{half} = 5$, where we will randomly pick just 1 context word at each side of the target. We will train the network for 10^4 iterations using Adam with the default parameters of the TensorFlow Class.

Figure 2 shows the loss during training. As we can observe in this example it seems clear that the network has actually learnt something. But since we don't have a measurable goal in this study, it is hard to tell from the loss if what the network is learning is useful or not.

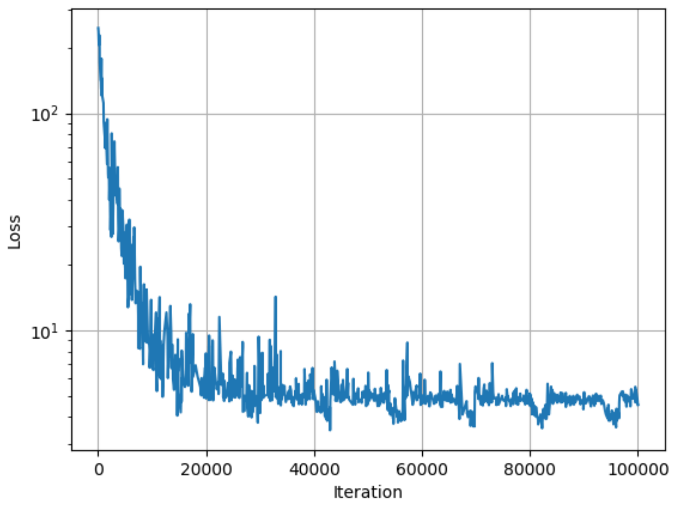

Figure 2: Evolution of the loss with training.

A qualitative way to know whether the network is actually learning or not, is looking at the representation of some random words during training. Table 1 shows through training the 8 closest words of some random examples in the embedding representation. As we can observe in this table, in the first step close words in the embedding space have apparently no relation with one another, but as we train, semantically similiar words get closer in the embedding space. After a lot of training, we can already see how the embedding relates shooting percentages (2-of-2 and 4-of-4 are together in the embedding space), player names are close to each other and the name of the days in a week got also grouped together.

Training Step 0
Nearest to win: affordable, rebounded, budget, pair, inevitably, 1990's, asleep, percentages,
Nearest to wednesday: curiosities, vacancy, manning, promised, blame, handicap, millsap's, garnett,
Nearest to at: kawhi's, sweeten, taurean, sooners, 20-34, others, creating, irate,
Nearest to should: one-hundred, unleashed, perturbed, 4-of-27, 2014-17, cling, evaluating, fat,
Nearest to mccaw: out-rebounded, doctor, re-evaluations, iv's, adjusts, upfront, 1-of-4, activities,
Nearest to 2-of-2: mclemore's, integrity, unselfishly, centerless, street, zones, aggressive, first-rounders,
Nearest to seems: dinwiddie's, credit, reacts, 787, shallow, clog, carrying, terry,
Nearest to tim: tantrum, dnp-illness, 102-100, porter-like, nuggets, news', 14-of-17, dove,
Training Step 20000
Nearest to win: pair, victory, loss, block, game, vs, of, year,
Nearest to wednesday: night, another, tuesday, friday, kept, saturday, appears, way,
Nearest to at: get, to, saturday's, set, friday's, expected, in, him,
Nearest to should: will, players, be, another, debut, after, late, role,
Nearest to mccaw: blazers, bender, plus, roll, might, monday, 1-of-4, 23,
Nearest to 2-of-2: 6-of-8, 1-of-3, added, four, 4-of-4, while, 0-of-4, six,
Nearest to seems: credit, dennis, terry, rough, still, probable, row, now,
Nearest to tim: nuggets, that, consistent, dealing, wade, back-to-back, adebayo, doesn't,
Training Step 90000
Nearest to win: loss, victory, game, but, the, he, to, with,
Nearest to wednesday: tuesday, friday, saturday, thursday, sunday, monday, night, to,
Nearest to at: in, but, to, a, the, and, with, of,
Nearest to should: could, will, to, may, he's, still, is, for,
Nearest to mccaw: nick, casspi, iguodala, draymond, livingston, javale, shaun, quinn,
Nearest to 2-of-2: 4-of-4, 6-of-8, 3-of-3, 1-of-3, 1-of-1, 5-of-5, 5-of-7, 2-of-3,
Nearest to seems: sounds, looks, dinwiddie's, burning, six, psycho, drastically, pointed,
Nearest to tim: hardaway, kanter, mcdermott, beasley, courtney, lee, thomas, jr,

Table 1: Nearest words in the embedding space for some example during training.

After the whole training we can see what words related to NBA basketball got closer to one another. In table 2 we show for some basketball related words, the nearest 8 examples found in the embedding space. As we can observe, even though we just trained the network with raw data, the embedding representations of semantically similar words are similar. For example, terms regarding actions of the game, the name of NBA teams, the name os some players and numbers are related to numbers (even smaller and larger numbers are close with each other). Note an interesting thing in the column with player names, all the players that appeared closer in the embedding space play for the Boston Celtics.

block	cavaliers	irving	23	1	3-of-3
blocks	heat	kyrie	24	2	4-of-4
steal	spurs	horford	32	3	2-of-2
turnover	trail	jaylen	28	7	5-of-6
turnovers	76ers	tatum	25	5	5-of-5
steals	orlando	rozier	20	6	2-of-3
assist	celtics	hayward	18	4	6-of-9
assists	knicks	baynes	22	0	5-of-7
rebound	pelicans	marcus	27	8	1-of-1

Table 2: Nearest words in the embedding space for some examples.

Following the results from the table before, we will try to visualize in a low dimensional space all players together with the team they play for. The visualization technique used in this experiment to represent high dimensional space vectors like the ones from the embedding space into only 2 dimensions is the so called t-Distributed Stochastic Neighbor Embedding (t-SNE). Figure 3 shows the representation of all the player occurrences for 5 different teams. As we can observe, we can clearly see five groups corresponding to every team represented. Note that different players from different teams may have the same surname and this can obviously lead to some errors. However, looking at the results there are some quite identifiable groups.

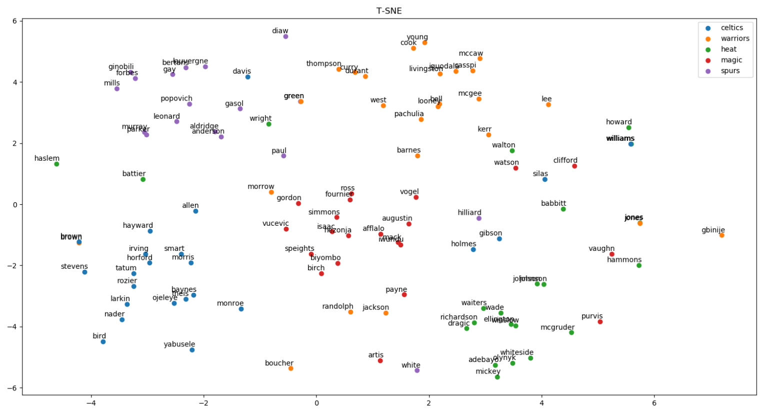

Figure 3: T-SNE representation for different players for five different teams

5 Conclusions

In this report, a word embedding was trained in a specific corpus of NBA basketball information. On the one hand, an implementation of the skip-gram word2vec model was programmed and trained in TensorFlow. On the other hand, an analysis of the resulting word embedding was performed, observing some of the interesting properties that these word transformations have.

We observed from the results obtained with the embedding that we can capture semantic relations between words just training the network with raw data, without any supervised information. This is a very interesting property that can be used in many NLP applications, using these vector representations of the words to train any Neural Network architecture in any specific task.

Initially, the idea was using these embedding representations in some simple task, in order to test the properties of this transformation in some quantitative way. However, the implementation of the word embedding took longer than expected and no more tests or further implementations were done. Future works extending this study could be then implementing some simple tasks with different NN architectures using the word embeddings obtained here.

References

[1] Rotoworld. (2017-2018). Player news, [Online]. Available: `http://www.rotoworld.com/playernews/`.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", Jan. 16, 2013. arXiv: `1301.3781v3 [cs.CL]`.

[3] TensorFlow. (). Word2vec tutorial, [Online]. Available: `https://www.tensorflow.org/tutorials/word2vec`.