A Bidirectional LSTM Model for Recalling Jokes Using Word Embeddings

Juan Lao Tebar

January 5, 2018

Abstract

In this practice we make some experiments with a model that memorizes question-answer jokes using word embeddings. We determine how the properties of the word embeddings affect to the performance of the model, and we analyze the behavior of the system when we play with some semantic regularities.

1 Introduction

Humor is the disposition of certain cognitive experiences to cause laughter and produce enjoyment, a complex process difficult to analyze from a scientific point of view. Many hypothesis try to explain what humor is, how it works, or what social function it serves. For example, Peter McGraw, 2012 [2] states that "humor only occurs when something seems wrong, unsettling, or threatening, but simultaneously seems okay, acceptable or safe".

Humor can be verbal, visual or physical but, no matter the form, there is a complex underlying relation between the elements that basically manifest a reflexion, an imitation, a surprise, a paradox, an ambiguity or an exaggeration. These concepts are presented with particular methods of different degrees of complexity, such as metaphors or hyperboles.

In particular, jokes are verbal representations of humor that, in the linguistic plane, consist of complex constructions that play not only with the objective meaning of the terms but also with the subjective interpretation and culture of the interlocutors.

Word embeddings are good representations of the semantic meaning of linguistic terms, associating words with descriptive numerical vectors. By defining a distance function between those vectors, we can find solid logical relations between them. [1]

In this practice we analyze the behavior of a model that uses word embeddings for recalling jokes.

Our model is a sequence-to-sequence bidirectional LSTM neural network trained with a Question-Answers Jokes dataset[1] consisting of 38,269 jokes from

[1]`https://www.kaggle.com/jiriroz/qa-jokes`

Reddit. The input of the model is a question and the output is a—relatively—funny answer. The main objective of our work is to play with different configurations of embedding sources and sizes in order to extract some conclusions about them, thus we do not focus on the details and fine-tuning of the network.

It is very difficult to generate new jokes, even as a human. Before starting working on this practice, we attempted to train the model with common cross-validation, but we found that it was not possible to decrease the validation loss at all, even after applying regularization methods. Our model can learn a lot of jokes from a particular training set, but it cannot generalize the *humor sense* to create new jokes. For this reason we focused this practice on teaching the model some jokes and playing with the semantic meaning of the questions and answers, checking what happens when we change the original words by similar alternatives, or what happens when we replace some words following a semantic regularity (e.g., changing the gender of the subject).

The word embeddings we use come from different sources: GloVe[2] [5], word2vec[3] [3] [4] and LexVec[4] [6]. Each provider offers different pre-trained versions, trained with different corpus and generating vectors of different dimensions.

This dataset contains jokes that can be considered inappropriate, irrespectful or offensive for certain collectives. We do not share the point of view of the people who created these jokes.

The code related to this work is public and available at github[5].

2 Report

2.1 Description of the Model

From the original dataset we use only jokes with a maximum question size of 19 words and a maximum answer size of 17 words.

Our model takes as an input a sequence of 19 embedded words and gives as output a sequence of 17 embedded words. Inputs and outputs with a lesser length are padded with a *padding token* vector until they have size 17. If a joke of the dataset contains a word that cannot be found in the embedding dictionary, the joke is discarded.

The neural network has a bidirectional LSTM layer of 128 units with an hyperbolic tangent activation for encoding the input, and another bidirectional LSTM of 128 units with also hyperbolic tangent activation for decoding the output. Finally, these hidden units are connected to a dense linear output layer.

We train the model with RMSprop, with a learning rate of 0.001, $\rho = 0.9$, $\varepsilon = 1 \cdot 10^{-8}$ and no learning rate decay.

As previously outlined, in this practice we do not validate or test our model. We just train it for 200 epochs and then we analyze the result. The loss is

[2] `https://nlp.stanford.edu/projects/glove/`

[3] `https://code.google.com/archive/p/word2vec/`

[4] `https://github.com/alexandres/lexvec`

[5] `https://github.com/juanlao7/reddit_qajokes-BLSTM`

calculated with the Mean Square Error (MSE), since we want to reduce the distance between our output and the target embedding vectors.

When the model gives an output in the form of a sequence of word embeddings, we can obtain a textual representation of the answer by finding the nearest neighbor of each vector. For this purpose we use an implementation of a K-D tree, where the cost of finding the nearest neighbors of a term is logarithmic.

2.2 Word Pre-processing

In order to train the model, we must first convert all the original jokes into sequences of embedded words, using the embedding dictionaries provided by each source of embedding vectors.

Unfortunately, the jokes from the dataset contain contractions ("what's", "father's", "you're"), words written in different cases ("What", "what", "WHAT") or symbols ("?", "!", ".", ","). In order to represent as many words as possible, we perform a pre-processing over each joke, that consists in the following:

We convert the original text into lowercase. We remove all the characters that are not numbers, letters or spaces. All the words in the joke—question and answer—must consist only of letters, numbers or start with a number and then follow with only letters (e.g., "18cm"). In other case, we discard the joke.

We apply the same process to the embedding dictionary, removing contractions and non-alphanumeric symbols from the keys and discarding those that do not meet the condition described in step 3.

With this process we aim to increase the size of the intersection between the vocabulary of the dataset and the chosen embedding set.

2.3 Padding Tokens

There are different ways of representing the *padding token*, directly affecting the performance of the model:

1. A vector where all components are zero.
2. A vector where each component i is equal to $m_i + 2 \cdot s_i$, where m_i is the mean value that component i takes among all the words of the embedding space and s_i is the standard deviation.
3. Same as the previous approach, but each component i is equal to $m_i + 3 \cdot s_i$.
4. We add a new extra component to all the embedding vectors. For the embedded words its value is zero, but for the padding token is 1; the rest of the components of the padding token remain as zero. In this case, when we need to obtain the textual representation of the output of our network, we define a threshold on the extra dimension to determine if an output is a padding token. If this threshold is not reached, then we obtain the nearest neighbor in the embedding space based on the original

vector dimensions. In our experiments we found that a close-to-optimal threshold that minimizes false positives and false negatives is 0.2.

In this experiment we test all the different ways of defining the padding token vector on GloVe—trained with 6 billion tokens, 300 dimensions—, and we compute the error of the models when determining if an embedded word is a padding token or not—number of false positives and false negatives.

Table 1 shows the average error when predicting the padding token on the first 50 answers of the dataset.

Encoding method	Error
0s	0.66 ± 1.51
Extra dimension	0.64 ± 1.49
2 standard deviations	0.76 ± 1.90
3 standard deviations	0.92 ± 2.39

Table 1: Average error when predicting the padding token on the first 50 answers of the dataset.

As we can see, the extra dimension method has the lowest error. In the following experiments we use this method for encoding the padding tokens.

2.4 Differences Between Embedding Sizes

When the training process in certain models takes too much memory or too much time to converge, we can consider using less dimensions to represent the embedded in exchange of losing representational power.

Our model does not suffer from any of these problems, but it is interesting to observe how the embedding size affects the results. In this experiment we test GloVe trained with 6 billion tokens with 50, 100, 200 and 300 dimensions.

As we can see in figure 1, more dimensions lead to a lower MSE. However, it is difficult to extract a conclusion from this fact, since the output of our network is an embedded word and comparing the MSE between models makes no sense. The embeddings have been trained using different neural networks, thus we can not guarantee that the embedding spaces they approximated have anything in common.

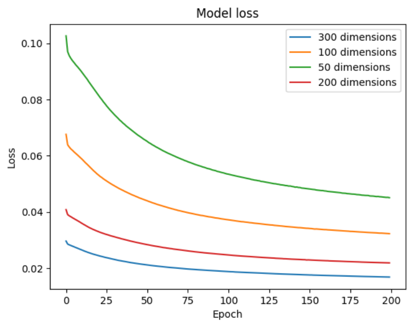

Figure 1: Training loss with each embedding size.

In order to analyze the results, we need to compare the textual representations of their output—number of correctly predicted tokens divided by 17, the answer length—. Table 2 shows the global accuracy of each approach on predicting the correct word.

Dimensions	Accuracy
50	0.7694 ± 0.1758
100	0.7688 ± 0.1734
200	0.7771 ± 0.1673
300	0.7829 ± 0.1690

Table 2: Textual representation accuracy for each embedding size.

2.5 Differences Between Corpus

GloVe word embeddings have been obtained from different corpus, resulting in a different number of words in each case. In particular, there is a package trained with 6 billion tokens extracted from Wikipedia 2014 + Gigaword 5 resulting in 400,000 words, and another trained with 42 billion tokens from Common Crawl resulting in 1.9 million words.

Increasing the number of words in the embedding space should not affect the loss of the model, but probably will decrease the accuracy of the textual representation, since the K-D tree will recall wrong neighbors that are very close to the correct term. In this experiment we train and compare the results of two models, one trained with the 400,000 word set and other with the 1.9 million.

Figure 2 shows the MSE evolution through the training, while table 3 shows the textual representation accuracy of each.

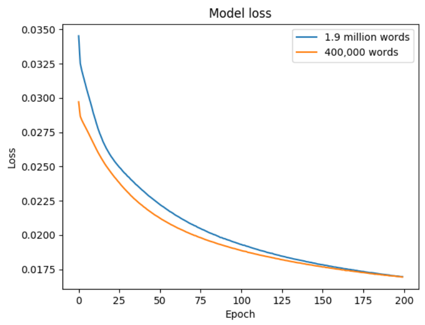

Figure 2: Training loss with each corpus.

Words	Accuracy
400,000	0.7829 ± 0.1690
1.9 million	0.7700 ± 0.1557

Table 3: Textual representation accuracy for each corpus.

Table 4 shows some examples where the model trained with the small set got better results than the bigger.

Question	Answer	400,000 words	1.9 million words
why was the router released early from prison?	it had connections	it **had** srivalo	it **was** assmtsale
why do black people like korean food?	because it has a little seoul in it	because it have a but bulletinyyy **in** it	because it was a though nowupload **both** it
what did the atheist fisherman say when asked about his catch?	there is no cod	**there is** but	**because it** even bmxznot

Table 4: Outputs of the model trained with each corpus.

2.6 Differences Between Embedding Algorithms

While GloVe creates its embeddings by aggregating global word-word co-occurrence statistics from a corpus, other algorithms such as word2vec implement continuous bag-of-words and skip-gram architectures for computing vector representations of words. Others such as LexVec use low-rank, weighted factorization of the Positive Point-wise Mutual Information matrix via stochastic gradient descent.

In this experiment we train our model with the following embeddings:

GloVe: Corpus from Common Crawl, 42 billion tokens, 1.9 million words, 300 dimensions.

word2vec: Corpus from Google News, 100 billion tokens, 3 million words, 300 dimensions.

LexVec: Corpus from Common Crawl, 58 billion tokens, 2 million words, 300 dimensions.

Figure 3 shows the MSE evolution through the training, while table 5 shows the textual representation accuracy of each.

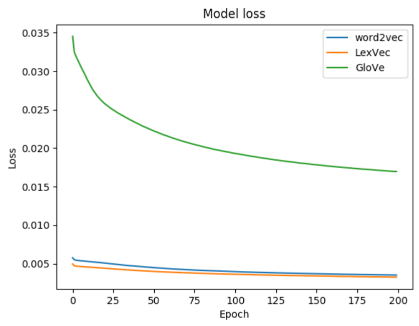

Figure 3: Training loss with each embedding embedding algorithm.

Algorithm	Accuracy
GloVe	0.7700 ± 0.1557
word2vec	0.7424 ± 0.1672
LexVec	0.7482 ± 0.1596

Table 5: Textual representation accuracy for each embedding algorithm.

As we can see, GloVe offers the best textual accuracy in our case.

2.7 Results

At this point we regret to say that our best model—trained with GloVe 6B 300—offers very poor results. Table 6 shows a variety of answers given by the network.

Question	Actual answer	Model prediction
why are frogs so happy?	they eat whatever bugs them	they they anyway httpwwwmediabynumberscom even
why did the hipster burn his tongue?	because he drank his coffee before it was cool	because he ate he instance before it was cool
whats brown and sticky?	a stick	a stick
why did mozart kill his chickens?	they were yelling bach bach bach bach	they were piyanart bulletinyyy bulletinyyy
which came first the chicken or the egg?	the rooster	the the bulletinyyy
who put semen in the basement?	i dont know thats just the way its spelled	i dont know oooooooooooooooooo–ooooooooooooooooooooo so indeed but but bulletinyyy
what do you do when your wife starts smoking?	slow down and apply lube	coming put and instance oooooooooooooooooo–ooooooooooooooooooooo
what is statistically three times worse than a war?	three wars	three piyanart
why dont drug addicts hang out at the beach?	they dont like getting sand in their crack	they dont so so bulletinyyy in their bulletinyyy
how does a jew make his coffee?	hebrews it	hebrews it

Table 6: Some model outputs.

As we can see, in some cases the model recalls the full answer to the question but in the majority of them it does not give even a syntactically correct output. Some words are incorrectly recalled with a high frequency (e.g. "bulletinyyy", "ooooooooooooooooooooooooooooooooooooooo"), probably because they are very close to the padding token.

One interesting detail we can observe is that the model usually outputs the right number of words, even if they are wrong. It seems that the network easily learns and focus on the number of padding tokens it must output at the end of the sentences.

2.8 Playing with Semantic Regularities

Our model *memorizes* the answers to the questions, but what happens if we replace a word in the question with a regularity? Does the model output an answer that also uses the regularity?

In GloVe—trained with 6 billion tokens, 300 dimensions—we can find several regularities, as shown in tables 7, 8 and 9.

hitler − germany + italy	hitler − germany + russia	hitler − germany + cuba
mussolini	stalin	cuba
hitler	hitler	fidel
italy	russia	castro
adolf	kremlin	cuban
fascist	putin	havana

Table 7: A regularity between dictators and the countries under their power.

merkel − germany + spain	merkel − germany + italy	merkel − germany + france
aznar	berlusconi	sarkozy
zapatero	prodi	chirac
sarkozy	merkel	merkel
merkel	sarkozy	fillon
spain	italy	france

Table 8: A regularity between presidents and the countries they preside.

dad − man + woman	uncle − man + woman	grandfather − man + woman
mom	aunt	grandmother
dad	niece	grandfather
mother	uncle	granddaughter
grandmother	grandmother	mother
widowed	mother	niece

Table 9: A regularity between family relationship and gender.

We tried to find jokes where we can apply those regularities, but in the majority of cases the model is unable to learn the answer, making impossible to apply them, as shown in table 10. In the case of the presidency regularity we could not find any joke in the dataset that involves merkel & germany, aznar & spain, berlusconi & italy or sarkozy & france.

Question	Actual answer	Model prediction
how did hitler originally get into power?	lets make germany great again	piyanart it but but
why did russia lose the race?	because it was stalin edit sorry communist jokes arent funny unless every gets them	because he was bulletinyyy oooooooooooooooooo– oooooooooooooooooooo bulletinyyy indeed bulletinyyy httpwwwmediabynumberscom indeed actually even piyanart so
why did russia sign the nonaggression pact with nazi germany?	they were stalin for time	they were bulletinyyy same but
why are cars in russia so slow?	because they are always stalin	because they are so
why didnt anyone drive stick in soviet russia?	they were afraid of stalin	they were indeed of

Table 10: All the jokes that contain hitler & germany, mussolini & italy, stalin & russia or castro & cuba. The model is unable to recall the important words.

Table 11 shows some cases where the model gives an answer where the regularity is applicable. This happens only on the gender and family relationship regularity.

Table 12 shows the results after replacing the question applying the regularity.

Question	Actual answer	Model prediction
whats the difference between me and your mom?	your mom hasnt had sex with your mom	your mom oooooooooooooooooo–ooooooooooooooooooooo when indeed while your bulletinyyy
whats the difference between your mom and your dad?	your dad still sucks dick	your dad piyanart bulletinyyy but actually
whats the difference between an airplane bathroom and your mom?	your mom can fit 3 people inside her comfortably	your mom so so but same same own

Table 11: Some jokes where the model successfully recalls words that involve a regularity.

Question	Expected answer	Model prediction
whats the difference between me and your mom?	your dad hasnt had sex with your dad	my piyanart oooooooooooooooooo–ooooooooooooooooooooo when piyanart but you
whats the difference between your mom and your dad?	your mom still sucks dick	my know actually but but but the piyanart
whats the difference between an airplane bathroom and your mom?	your dad can fit 3 people inside her comfortably	your actually so come actually bulletinyyy even

Table 12: Model output after applying the regularity.

As we can see, the model is unable to follow the regularity.

3 Discussion of the Results

As shown in the previous experiments, the model is not only unable to generalize the *humor sense* behind the jokes but also has a very low accuracy when trying to recall the ones used for training. Even so, we can observe some interesting details.

With regard to the representation of the padding token, we found that one good approach consists in adding a new extra dimension, setting it to 1 in the padding token, and to zero in the enbedded words. Then we define a threshold of 0.2 for determining if a vector is a padding token or not.

Other methods—such as defining the padding token as a vector of zero values or trying to *separate* it from the embedded words—lead to worse results, caused by the low precision of the model, that makes the K-D tree unable to recall the padding token properly.

When we compared the results of training our model with different embedding sizes—in terms of dimensions—, we confirmed that bigger vectors lead to a better textual accuracy in our model, since they are able to represent better the semantic differences between the embedded terms, increasing the distance of the words in the embedding space and decreasing the error of our recalling approach.

Training our network with a bigger corpus does not affect the MSE of the model; however, it slightly affects its textual accuracy. We found that a corpus of 400,000 words offers better results than a corpus of 1.9 million words. We think that the reason behind this behavior is just that the network has a low precision and the K-D tree fails to recall the correct word when there is a higher concentration of them in the embedding space.

When we tested different embedding algorithms we found that with GloVe we obtain a better textual accuracy than with LexVec, both trained with the same corpus and approximately the same number of words—1.9 million vs. 2 million—. Word2vec obtained the worst mark, but it has been trained with a different corpus and has 3 million words, so we cannot assure that the problem is in the algorithm; as shown before, a higher number of words in the embedding space lead to lower accuracies.

Finally, when we played with some regularities, we found that the model is not able to follow them.

4 Conclusion and Future Work

In this practice we analyzed the behavior of a model that uses word embeddings for recalling jokes.

The model cannot generate new jokes. It is only capable of *memorizing* the training set—with a low recall—and does not follow regularities. However, we observed some interesting details, commented in the previous section.

Some things we did not try but would be interesting to test are the following:

- Perform all the experiments again, but with a smaller training dataset, in order to increase the recall rate of the model.
- Test a more complex model, with a higher number of hidden layers not only in the encoder architecture but also in the decoder.

- Check what happens when each word of a question is replaced by the most close word in the embedding space.
- Check what happens when each word of a question is replaced by a synonym, not taking into account the position in the embedding space.

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[2] A Peter McGraw, Caleb Warren, Lawrence E Williams, and Bridget Leonard. Too close for comfort, or too far to care? finding humor in distant tragedies and close mishaps. *Psychological Science*, 23(10):1215–1223, 2012.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[5] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[6] Alexandre Salle, Marco Idiart, and Aline Villavicencio. Matrix factorization using window sampling and negative sampling for improved word representations. *arXiv preprint arXiv:1606.00819*, 2016.