Language Models: investigating the domain dependencies of learnt word embeddings.

Amedeo Cuttica - Università degli Studi di Torino

***Abstract*— We give a general description of the problem of efficiently representing words when applying machine learning algorithms to textual data. A widely successful way to address this has proved to be *pre-train* word embeddings while training a Statistical Language Model. Our goal is to train the same recurrent *left-to-right* language model over data taken from different domains and, subsequently, investigate to what extent the relations between learnt word representation depend on such domain.**

I. Introduction

When performing some sort of statistical analysis on textual data, a way to represent sentences in a mathematically tractable form is required. A common, although not unique, solution is to represent words via numerical vectors: these easily lend themselves to math operations and have an easy interpretation ad feature vectors when it comes to Machine Learning algorithms [1].

What kind of information should these vectors encode? This clearly depends on the goal of the specific task one wishes to perform. For example, a sentiment analysis task may require different information compared to, say, a part-of-speech tagging task. The major approach has hence for long been for the researchers themselves to extract complex task-specific sets of features from the data and then feed such features to usual ML systems. This feature-engineering process is a completely empirical procedure based mainly on intuition, trial and error [2].

In order to address this issue, a system that automatically learns (continuous) representations for each word that are relevant for the task in question would be highly desirable. Note that these learnt features could easily be augmented with any desired *ad hoc* feature that is considered to be useful for the task in question.

II. Solution

A natural approach aimed at avoiding the mentioned feature-engineering procedure could be the following:

- Represent each word with as little prior knowledge as possible: as indices in the dictionary.
- Project such indices on a suitable *embedding* space $\mathbb{R}^d$ via a matrix multiplication/lookup table operation.
- Allow the parameters of the above projection to be learnt, together with all the other parameters of the system, during the task-specific training.

Despite being simple and elegant, this approach typically fails in practice and the reason is trivial: given a dictionary of size D, which is usually of the order of magnitude of 100.000, the above projection matrix consists of $D \times d$ parameters. The labeled data available for task-specific training is almost always not enough to successfully train such a huge matrix: if we would prefer semantically and syntactically similar words to be *close* in the embedding space and this is typically not the case, due to lack o sufficient training data.

Different methods have therefore been proposed in order to overcome the problem. Most of them consist in training the embeddings while training a Statistical Language Model and then use the learnt word representations as a starting point in the task-specific system (Transfer Learning). For this reason one typically speaks about *pre-trained* word embeddings. The advantage of Language Models is that they can be trained on unlabeled data, raw text in this case, which is abundant and easily accessible. Moreover, they proved themselves to be able to learn non-trivial semantic and syntactic regularities between words in the dictionary [6].

Stating here that the possibilities are almost unlimited when it comes to Language Models and Transfer Learning, we now present a simple *left-to-right* model and its training over different unlabeled text corpuses.

III. Model

Our model is a simple single layer *Elman* recurrent neural network with a learnable embedding layer on top of it. This is almost identical to the model presented by

Tomas Mikolov in [5], with the only difference being an additional linear projection between the embedding layer and the recurrent layer. Specifically, the output is computed as follows:

$$
\begin{aligned}
w_t &= E(i_t) \\
s_t &= f(Aw_t + a + Bs_{t-1} + b) \\
y_t &= g(Cs_t)
\end{aligned}
$$

where i_t represents the index in the dictionary of the input word at time step t, f is a *hyperbolic tangent* squashing function, C is a projection again over the dictionary and g is a *log-softmax* function.

By interpreting the output y_t as log-probabilities for the next word i_{t+1}, we can train the model to predict the next word given the previous ones. The advantage of a recurrent model consists in the fact that information can cycle inside the system for an arbitrarily long time so that it is able to learn arbitrarily long dependencies between terms in the input sequence (at least in theory [3]). In contrast, when using a feed-forward model [4] the length of the conditioning context is fixed and has to be specified as a hyperparameter.

In particular, we use and embedding size d and a hidden size h both equal to 128 (not necessary for them to be the same). These are the only hyperparameters that need to be set when building the model, together with the dictionary size which in turn depends on the data.

IV. Data

We collected three different datasets for our Language Model training. The first is a reduced version (around 1.000.000 words) of the renowned *Penn Treebank* corpus [7], a dataset widely used for several *NLP* training tasks. We used The Penn Treebank corpus in order to first train the model on a "neutral" domain.

Then we collected tweets from two different datasets originally created for Sentiment Analysis [9] and Offensive Language Detection [8] (classification) respectively. From the first we extracted only the tweets annotated as *"racist/sexist"*, while from the second only the ones annotated as *"hate speech"* or *"offensive language"*. On the resulting corpus of tweets (around 300.000 words in total) we performed some text cleaning: the goal was to discard rare or special words as, for example, hashtags, usernames, hyperlinks, numbers and emoticons. Indeed, such words would only result in enlarging the size of the dictionary and, consequently, of our model too.

Putting together all the different words encountered in the three (cleaned) datasets, we obtain a dictionary consisting in around 27.000 words.

V. Training

The model is trained via Truncated Backpropagation Through Time TBPTT: indeed, it would be unfeasible to store outputs and losses for the whole training set. Moreover, when backpropagating the error on an overly long sequence it is very likely to incur in *vanishing/exploding gradient* problems.

In practice the algorithm works as follows: the training sequence of words is divided into several input subsequences of equal length k, each paired with its corresponding target subsequence. Since we are training the model to predict the next word given the previous ones, the target subsequence is simply the input subsequence shifted forward by one word.

$$
\begin{aligned}
input &= (i_1 \dots i_k) \\
target &= (i_2 \dots i_{1+k})
\end{aligned}
$$

Then, for each input sequence the model produces an output sequence of the same length: each element represents a log-probability distribution and it is used to calculate the *negative log-likelihood* of the corresponding target word. Finally, the loss on the subsequence is computed as the average of the resulting k negative log-likelihoods.

$$
\begin{aligned}
output &= (y_1 \dots y_{1+k}) \\
loss &= \frac{1}{k} \sum_{j=1}^{k} -logp(i_{j+1}|y_j)
\end{aligned}
$$

The obtained loss is then backpropagated and the parameters of the model are updated according to the computed gradients via usual gradient descent with a learning rate equal to 0.1.

The above procedure is repeated for each subsequence until the whole training data has been processed, that is, until one training epoch is completed.

After each training epoch the model is validated on a validation set. When the negative log-likelihood on the validation set, computed as above and then averaged again over all the subsequences, starts growing, possibly indicating overfitting, we stop training. In any case, we stop training when 20 epochs are completed.

VI. Results

During training we clearly notice the difficulty of learning word representations: the weights of the embedding matrix get updated way more slowly than the rest of the parameters of the model. As a consequence, even if the loss on the validations set gradually decreases, we don't really witness sensible changes in the embedding space. In practice, we investigate *neighbours* of specific words in the embedding space, where the closeness is computed via cosine similarity: even after several training epochs these appear to be resistant to changes. This is true for both the training on the Penn Treebank dataset and the training on the Twitter dataset.

A possible reason could, again, be the insufficiency of training data. The embedding matrix contains more than $3.5M$ weights, a number definitely too large to successfully train over our reduced dataset.

This problem can be dealt with in two ways. The first is, trivially, to enlarge the available training corpuses. The second, in our opinion more relevant, is to put more effort in the data preprocessing procedure. Stemming, lemmatization and the removal of additional rare words could bring some important reduction in the vocabulary size and, in turn, in the size of the embedding matrix.

References

[1] *Word Embeddings: A Survey*, Felipe Almeida and Geraldo Xexeo, 2019.

[2] *Natural Language Processing (almost) from Scratch* , Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa, 2011.

[3] *Learning long-term dependencies with gradient descent is difficult*, Y. Bengio, P. Simard and P. Frasconi, 1994.

[4] *A Neural Probabilistic Language Model*, Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin, 2003.

[5] *Recurrent neural network based language model*, Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernock and Sanjeev Khudanpur, 2010

[6] *Linguistic Regularities in Continuous Space Word Representations*, Tomas Mikolov, Wen-tau Yih and Geoffrey Zweig, 2013.

[7] *Building a Large Annotated Corpus of English: The Penn Treebank*, Mitchell P. Marcus, Mary Ann Marcinkiewicz and Beatrice Santorini, 1993.

[8] *Automated Hate Speech Detection and the Problem of Offensive Language*, Thomas Davidson, Dana Warmsley, Michael Macy and Ingmar Weber, 2017.

[9] https://www.kaggle.com/arkhoshghalb/twitter-sentiment-analysis-hatred-speech