Replication of “Hash Embeddings for Efficient Word Representations”

Pravar D.Mahajan [*]
Department of Computer Science
The Ohio State University
Columbus, OH 43202
`mahajan.89@osu.edu`

Abstract

As part of the of Nurture.AI’s Global NIPS Implementation Challenge initiative, we are replicating and verifying the work of the NIPS ’17 paper “Hash Embeddings for Efficient Word Representations”. This paper explores the efficacy of hash embedding technique by evaluating performance of classifier models on various hash-embedded documents. We are able to closely replicate the results of the original paper and also notice that the new proposed embedding model works faster than the usual embedding model in training.

1 Introduction

An important facet of science is replication, that is, being able to reproduce the results claimed in any scientific work. Independent testing, challenging and verification of published results not only add to the credibility of the research, but also helps in democratizing scientific knowledge. Research based on machine learning should therefore be put up against the same verification process in order to be called science.

However, there are a few challenges in the domain of machine learning which makes it difficult to reproduce the results exactly. The first and foremost challenge is the unavailability of data. The bigger reason, however, is that there are several tricks which go into the implementation of these Machine Learning models - like hyperparameter tuning, validation splits, optimization algorithms and initial parameters, which are often skipped or stated casually in these papers. As a result, beginners in the area of Machine Learning often find it challenging to implement the models presented in the published papers.

As part of Global NIPS Implementation Challenge by Nurture.AI, we are replicating the results of the paper - “Hash Embeddings for Efficient Word Representations” [1]. The code to replicate has been released in the public domain [2]. The outline of this report is as follows - In the next section, an outline of the paper is provided. In the next sections we describe our replication methodology and finally summarize the results.

2 Outline of the Paper

While applying deep learning to NLP tasks, the words are converted into their respective embedded representations. However, embedding matrices are expensive to store and even to learn, since they

[*]Many thanks to Dan Svenstrup, Jonas Meinertz Hansen and Ole Winther (Technical University of Denmark), the authors of the original work

[2]https://github.com/pravarmahajan/Nips-Paper-Summaries/tree/master/hash-embeddings-for-efficient-word-representations

Table 1: Description of Datasets Used

	#Train	#Test	#Classes	Task
AG's news	120k	7.6k	4	English News Categorization
DBPedia	450k	70k	14	Ontology Classification
Yelp Review Polarity	560k	38k	2	Sentiment Analysis
Yelp Review Full	560k	50k	5	Sentiment Analysis
Yahoo! Answers	650k	60k	10	Topic Classification
Amazon Review Full	3000k	650k	5	Sentiment Analysis
Amazon Review Polarity	3600k	400k	2	Sentiment Analysis

Table 2: Ngram sizes for different datasets

Dataset	n
AG's News	10
DBPedia	10
Yelp Review Polarity	6
Yelp Review Full	6
Yahoo! Answers	6
Amazon Review Full	2
Amazon Review Polarity	1

contain a lot of parameters (vocab size $\times$ embedding dimension). This problem becomes more apparent as we move to higher ngrams as unit tokens.
To deal with this issue of high-dimensionality for higher ngrams, *feature hashing* is used to allocate each word to one of the discrete "buckets" $\{1, 2, \dots B\}$. Since B << size of the token space, multiple tokens end up having same vector representations. Further, owing to its discrete nature, it is not easy to backpropagate the loss via gradient decent to learn a hashing in which similar tokens get similar buckets.
This paper proposes an interpolation between the standard embedding and the feature hashing method. As in feature hashing, each token is assigned one of the B buckets. Each bucket maps to a continuous vector from a pool of embedded vectors, which can be learned just like the standard embedding. For more sophisticated models, number of hashing functions may be increased for better token representations. These multiple vectors corresponding to different hashing functions may be aggregated by concatenation or summing (as done in this paper).
The evaluation of their proposed model is done on 7 different datasets 1. The replication results may be found in 3. Please refer to Table 3 of [1] for a comparison of the results to other embedding methods.

3 Methodology

Since our focus is to replicate the results of the experiments performed in the work, we used the architecture as described in Section 5.2, 5.3 and 5.4 of the paper directly. For preprocessing, we replaced all the punctuations with spaces and converted all characters to lowercase. We used early stopping with patience = 1, since patience of 10 produces significantly lower accuracy. Due to limited computational resources, we could not exactly produce all n-grams for n < 10 for all the datasets. Therefore, we have limited our n-gram sizes to different values depending on the dataset size, which has been summarized in 2. As stated in the paper, higher order n-grams would be infrequent, therefore this should not make a big difference to our results. As described in the paper, we compare the results of experimenting with the following combination of model variations:

- Hash Embedding vs Standard Embedding
- No dictionary vs Precomputed Dictionary of ngrams

The results have been summarized in 3

Table 3: Summary of Results

dataset	hash emb no dict	std emb no dict	hash emb w dict	std emb w dict
AG News	91.13	91.18	91.13	91.32
Amazon Full	56.04	57.99		
Amazon Polarity	91.97	91.96		
Dbpedia	98.37	98.40	98.30	98.15
Yahoo QA	69.96	71.03		
Yelp Full	56.50	55.78		
Yelp Polarity	90.59	90.23		

4 Results and Discussion

Table 3 summarizes the results we obtained. Although we couldn't replicate the numbers *exactly*, the accuracy numbers are somewhat close. We attribute the replication differences to:

- Randomness in train-test split and order of training
- Approximation by not including higher order n-grams, as described in 2
- Certain assumptions which were not clear in the paper (what constitutes a punctuation, whether punctuations are replaced with space or simply removed, decapitalization, etc.)

In spite of the differences observed, we notice a strong correlation with the results presented in the paper. We also notice that hash embeddings indeed lead to faster training. For example, we notice that the training was 2-3 times faster for hash embedding compared to standard embedding, with insignificant drop in the accuracy, on every dataset. This is in line with the claim of the paper that the newly proposed embedding scheme allows for inexpensive embedding computation at the cost of marginal to no drop in accuracy.

References

[1] Dan Tito Svenstrup, Jonas Hansen, and Ole Winther. Hash embeddings for efficient word representations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4935–4943. Curran Associates, Inc., 2017.