Comparative Analysis: Dynamic Embedding Techniques Versus Word2Vec for downstream tasks

Rushank Mihir Shah, Priyam Vora, Devika Bhavesh Patadia, Harsh Chheda, Sneh Samir Shah
[rushankm, priyamvo, dpatadia, harshchh, snehsami]@usc.edu
University of Southern California

Abstract

This report introduces an innovative approach to deploying NLP models efficiently on devices with limited computational resources. By comparing the performance and efficiency of dynamic rolling hash embedding with traditional Word2Vec embeddings, we aim to demonstrate our proposed model's feasibility, especially for smartphones and smartwatch applications. Our findings, based on experiments with the Amazon Reviews dataset for sentiment analysis, reveal that our model achieves comparable accuracy with significantly reduced model size and processing time.

1 Introduction

Natural Language Processing (NLP) has surged with transformer models like BERT, yet their size restricts deployment to well-resourced environments. EELBERT (Embeddingless BERT) offers a solution by compressing models while maintaining performance integrity, inspiring our work. We explore a dynamic rolling hash algorithm for efficient word embedding generation, uniquely capable of context-aware embeddings on the fly, reducing computational resources and model size.

Our approach aims to democratize advanced NLP by enabling deployment on resource-constrained devices. Comparing our dynamic rolling hash embeddings with Word2Vec, we aim to match or surpass performance in tasks like sentiment analysis, with significantly reduced resource demands.

This analysis not only shows the feasibility of deploying advanced NLP on less capable devices but also highlights practical benefits in real-world applications. By leveraging dynamic embeddings inspired by EELBERT, our work advances the accessibility of state-of-the-art NLP technologies across diverse devices and users.

2 Related Work

The challenge of optimizing memory and performance in large NLP models has been the focus of extensive research. This section reviews vital strategies that address the efficiency of these models, especially in the context of embedding size reduction and innovative embedding techniques.
Reducing Embedding Sizes: Efforts focus on shrinking token embeddings, mapping input tokens to real-valued vectors. Studies include Lebret and Collobert's (Lebret and Collobert, 2014) use of Principal Component Analysis (PCA) to reduce word embedding dimensionality. Other methods like LASPE (Arora et al., 2018) and Word2Bits (Lam, 2018) compress GloVe and Word2Vec embeddings, respectively, using techniques like matrix factorization and quantization.

Hash and Randomized Embeddings: Tito Svenstrup et al. (Svenstrup et al., 2017) explore hash and randomized embeddings, offering a means to handle large vocabularies with lower memory overhead. Dynamic and Projection Embeddings: Ravi and Kozareva (Ravi and Kozareva, 2018) and Ravi (Ravi, 2017) introduce projection networks for generating embedding vectors directly from text, eliminating the need for a traditional embedding layer. Subsequently, Sankar et al. (Sankar and Ravi, 2021) and Ravi and Kozareva (Sankar et al., 2021) extend this to word-level embeddings using LSH-based techniques. EELBERT (Cohn et al., 2023) dynamically generates embeddings during runtime, reducing memory footprint without sacrificing task accuracy. Our project builds on EELBERT's approach, exploring dynamic embeddings further with a rolling hash algorithm to enhance model compression and efficiency.

3 Problem Description

In the rapidly advancing field of natural language processing (NLP), deploying sophisticated mod-

els like BERT on devices with limited resources, such as smartphones and smartwatches, poses significant challenges. These models require substantial memory and processing power, which is not always available on smaller devices. Traditional embedding techniques, while effective, are often too resource-intensive for such environments.

Our project aims to overcome these constraints by developing a novel solution that enables the use of advanced NLP technologies on a broad range of devices without compromising accuracy. We are exploring dynamically generated word embeddings through a rolling hash algorithm, offering a more compact and efficient alternative for creating context-aware word embeddings. This approach could potentially maintain or even enhance the performance of existing models like Google's Word2Vec and custom Word2Vec models built with the GenSim package, while drastically reducing memory and processing requirements.

Using the Amazon Reviews dataset, we conduct sentiment analysis in both binary (positive vs. negative) and ternary (positive, negative, neutral) formats to evaluate the effectiveness of our method. Our ultimate goal is to develop a smaller, lighter model that delivers performance comparable to or better than traditional models, thereby extending NLP applications to more devices and making cutting-edge technology more accessible for everyday use.

4 Methodology

The methods employed include utilizing specific materials such as a designated corpus of language data along with any existing data annotations. For instance, the corpus utilized in this study is the Amazon Reviews Dataset, comprising 2,640,254 reviews spread across 15 columns, providing a substantial source of real-world textual data. In terms of data selection, subsets were chosen for sentiment analysis classification, with 200,000 reviews for training and 40,000 for testing in binary classification (positive and negative reviews), and 250,000 reviews for training and 50,000 for testing in ternary classification (positive, negative, and neutral reviews). The Amazon Reviews dataset already includes pre-existing sentiment labels, categorizing each review as positive, negative, or neutral, which facilitates supervised learning tasks by offering clear target outcomes for model training and validation.

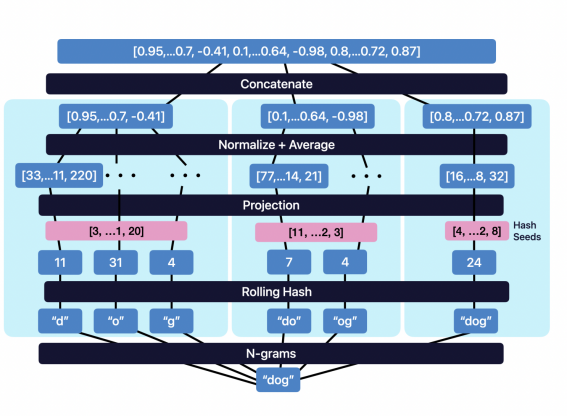


Figure 1: Computing Dynamic Hash Embeddings

Regarding data processing, the procedure involved various steps and algorithms. Initially, the dataset underwent preprocessing by selecting pertinent columns, addressing missing values, and converting the rating column into a numeric format. Sentiment labeling was then conducted based on ratings, with ratings above 3 considered positive, those less than or equal to 2 deemed unfavorable, and those equal to 3 labeled as neutral. Each rating category was subsequently downsized to 50,000 samples to counteract class imbalance. Text data preprocessing included converting text to lowercase, removing HTML tags, URLs, and special characters, as well as normalizing whitespace. Furthermore, contractions were expanded to ensure consistency in text representation. These preprocessing steps were integral in readying the dataset for accurate sentiment analysis, enabling the extraction of meaningful insights from the reviews.

Our n-gram pooling hash function methodology is shown in Figure 1, with operations in black boxes and black lines going from the input to the output of those operations. Input and output values are boxed in blue. For ease of notation, we refer to the n-grams of length i as i-grams, where i = 1, ..., N, and N is the maximum n-gram size. The steps of the algorithm are as follows:

1. Initialize random hash seeds (h): There are d hash seeds, where d is the size of the embedding we wish to obtain. The d hash seeds are generated via a fixed random state, so we only need to save a single integer specifying the random state.

2. Hash i-grams to get i-gram signatures (s_i): There are $k_i = l - i + 1$ number of i-grams, where l is the length of the token. Using a

rolling hash function (Wikipedia contributors, 2023), we compute the i-gram signature vectors.

3. Compute projection matrix for i-grams: For each i, we compute a projection matrix P_i using a subset of the hash seeds. The hash seed vector h is partitioned into N vectors, boxed in pink in the diagram. Each partition h_i is of length d_i, where $\sum_{i=1}^{N} d_i = d$, with larger values of i corresponding to a larger d_i. Given the hash seed vector h_i and the i-gram signature vector s_i, the projection matrix P_i is the outer product $s_i \times h_i$. To ensure that the matrix values are bounded between $[-1, 1]$, we perform a sequence of transformations on P_i:

$$P_i = P_i \% B$$

$$P_i = P_i - (P_i > \frac{B}{2}) \times B$$

$$P_i = P_i / \left(\frac{B}{2}\right)$$

where B is our bucket size (scalar).

4. Compute embedding, e_i, for each i-gram: We obtain e_i by averaging P_i across its k_i rows to produce a single d_i-dimensional vector.

5. Concatenate e_i to get token embedding e: We concatenate the N vectors $\{e_i\}$ where $i = 1$ to N to get the token's final embedding vector, e of size d.

For a fixed embedding size d, the tunable hyperparameters of this algorithm are N, B, and the choice of the hashing function. We used $N = 3$, $B = 10^9 + 7$, and the rolling hash function.

5 Model Architecture

The model architecture comprises a Feedforward Neural Network (FFNN) and a Convolutional Neural Network (CNN) designed for sentiment analysis. The FFNN, named FFNetBinary, consists of three fully connected layers with 50, 10, and 2 or 3 units, respectively. The input layer has 300 units corresponding to the size of the word embeddings. Each hidden layer is followed by a Rectified Linear Unit (ReLU) activation function and dropout with a probability of 0.2 to prevent overfitting. The CNN employs two convolutional layers with 50 and 10 output channels, respectively, followed by max-pooling layers. The convolutional layers use a kernel size of 3 and padding of 1 to maintain the input size. The output of the convolutional layers is flattened and passed through a fully connected layer with 120 units and a ReLU activation function. Dropout with a probability of 0.3 is applied before and after the convolutional layers to regularize the network. For ternary classification, the only modification is the presence of three output neurons in the last layer instead of two, enabling classification into positive, neutral, and negative sentiments. These model architectures are designed to learn and classify sentiment from the preprocessed text data effectively.

For this purpose, we use the average word embedding vectors for each review as the input feature:

$$X = \frac{1}{N} \sum_{i=1}^{N} W_i,$$

where W_i represents the word embedding for each word in a review, and x is the input vector representing a review.

6 Experimental Results

6.1 Datasets

The dataset comprises a substantial collection of 2,640,254 Amazon reviews, meticulously partitioned into training and testing subsets. For binary and ternary classifications, the training sizes were set at 200,000 and 250,000 reviews, respectively, with corresponding testing sizes of 40,000 and 50,000. We split the dataset into 60% training, 20% validation, and 20% testing data. The `batch_size` has been set to 100.

6.2 Baseline Methods

The Google model used for this task is trained on the Google News dataset. The baseline models utilized specific hyperparameters tailored to their tasks. For binary classification using FFNN, 50 epochs were employed with a learning rate of 0.007, while ternary classification with FFNN utilized 50 epochs and a learning rate of 0.0007. Binary classification with CNN was conducted over 10 epochs with a learning rate of 0.005, and ternary classification with CNN employed the same number of epochs but with a slightly higher learning rate of 0.005.

Model Type	Google Model	Word2Vec	Proposed Model	% Change Google	% Change Word2Vec
Binary FNN	124 KB	124 KB	63.6 KB	-48.70%	-48.70%
Ternary FNN	124 KB	124 KB	63.7 KB	-48.62%	-48.62%
Binary CNN	2.7 MB	2.7 MB	185.4 KB	-93.95%	-93.95%
Ternary CNN	2.7 MB	2.7 MB	190 KB	-92.9%	-92.9%

Table 1: Comparison of Model Sizes and Percentage Change

6.3 Evaluation Protocols

The models using dynamic hashing-based techniques differed from baseline models as follows: binary classification with FFNN utilized 75 epochs and a learning rate of 0.003, while ternary classification with FFNN involved 75 epochs and a learning rate of 0.05. For CNN-based approaches, binary classification entailed 75 epochs and a learning rate of 0.009, whereas ternary classification employed 30 epochs with a higher learning rate of 0.1.

6.4 Performance Analysis

Throughout our experimentation, the performance of various models was meticulously scrutinized. The Google Model exhibited accuracy rates ranging from 68.44% to 85.48% across different tasks and classifications, while the Word2Vec model showcased accuracy levels from 71.816% to 87.8325%. Our initial iteration of the proposed model yielded accuracy percentages between 63.0456% and 83.0975%. However, post-tuning, the proposed model demonstrated improved accuracy, achieving percentages ranging from 68.5149% to 87.68%.

6.5 Model Size Comparison

Regarding model size, the proposed models demonstrated significant reductions compared to the Google Model and Word2Vec. In Table 1, the proposed Binary FNN model reduced in size from 124 KB to 63.6 KB, representing a reduction of 48.70% compared to both the Google Model and Word2Vec. Similarly, the Binary CNN model reduced from 2.7 MB to 185.4 KB, marking a decrease of 93.95% compared to both baseline models. Similar compression was observed for ternary models as well

6.6 Model Time Comparison

Regarding computational time, the proposed models exhibited variations compared to baseline models. For instance, the Binary FNN model increased from 10 seconds to 14 seconds post-tuning, representing a 40% increase compared to both baseline models. Similarly, the Ternary CNN model increased from 24 seconds to 33 seconds post-tuning, reflecting a 37.5% increase compared to both baseline models. The rise in time is anticipated as we generate embeddings in real-time. These metrics offer a comprehensive understanding of model efficiency beyond accuracy analysis.

7 Conclusion

This study demonstrates the efficacy of dynamic rolling hash embeddings for deploying NLP models on resource-constrained devices, providing an alternative to traditional Word2Vec embeddings. Our tests using the Amazon Reviews dataset show that dynamic embeddings can match or exceed traditional methods in performance while significantly reducing model size and processing times. Inspired by embedding compression advancements, our technique allows for adaptive, efficient real-time text processing on mobile devices by generating context-aware embeddings on the fly. This reduces the computational load, essential for applications in environments with limited resources but needing advanced text analysis. Comparative analysis shows dynamic embeddings reduce model size by up to 48.70% and CNN model sizes by 93.95%, maintaining competitive accuracies, thus facilitating broader deployment and democratizing access to advanced NLP technologies.

8 Future Work

Several promising directions exist to improve dynamic rolling hash embeddings. Optimizing hash function parameters and exploring different hash types could enhance the robustness and accuracy of the embeddings. Experimenting with various hash table sizes or modifying the rolling mechanism may better adapt to linguistic contexts. Additionally, testing more NLP tasks like machine translation, named entity recognition, and summarization could assess the technique's broader applicability, especially in resource-limited settings. Integrating dynamic embeddings with edge computing could

reduce latency and bandwidth in real-time applications, such as interactive voice systems. Finally, user studies could reveal the practical impact and guide model refinements for real-world use.

9 Division of Labor Between the Teammates

9.1 Harsh Chheda

Conducted a comprehensive online search and review of approximately 10 research papers to deepen his understanding of various word embedding generation techniques. Analyzed methodologies used by researchers to determine which could be feasibly implemented within the project's timeline. Evaluated the strengths and weaknesses of different word embedding techniques, aiding in the selection process for our project's approach.

9.2 Sneh Shah

Dedicated significant effort to exploring numerous research papers on word embedding generation techniques, reviewing around 10 papers. Focused on identifying practical and innovative methods that could be implemented effectively within our project's constraints. Synthesized the findings from these papers to pinpoint common trends and novel approaches in word embedding generation, shaping the project's direction.

9.3 Priyam Vora

Played a crucial role in developing and implementing the dynamic hashing algorithm. Managed the data preprocessing tasks for the sentiment analysis component, integrating these processes with the dynamic hashing method. Responsible for generating embeddings and conducting thorough performance evaluations on various models to assess the efficacy of the dynamic embeddings.

9.4 Rushank Shah

Was instrumental in the development and implementation of the dynamic hashing algorithm. Took a significant role in integrating various components with the dynamic hashing approach and in the performance evaluations of the models. Assisted in documenting the project methodology and addressed various developmental challenges, ensuring smooth progression in project milestones.

9.5 Devika Patadia

Made substantial contributions to optimizing the dynamic hashing technique and was actively involved in generating embeddings. Led the training and testing phases of the dynamically generated word embeddings and performed comparative analyses with traditional Word2Vec embeddings. Handled the documentation of the project's status reports, detailing the tasks undertaken, methodologies used, and challenges encountered. Organized and facilitated regular team meetings and progress updates, maintaining effective communication and project coherence.

References

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2018. A simple but tough-to-beat baseline for sentence embeddings. In *International Conference on Learning Representations (ICLR)*.

Gabrielle Cohn, Rishika Agarwal, Deepanshu Gupta, and Siddharth Patwardhan. 2023. Eelbert: Tiny models through dynamic embeddings.

Maximilian Lam. 2018. Word2bits - quantized word vectors. https://github.com/MaximilianLam/Word2Bits.

Rémi Lebret and Ronan Collobert. 2014. Word representations via gaussian embedding. *International Conference on Learning Representations (ICLR)*.

Sujith Ravi. 2017. Projectionnet: Learning efficient on-device deep networks using neural projections. *arXiv*, abs/1708.00630.

Sujith Ravi and Zornitsa Kozareva. 2018. Projection-based embeddings for text classification. *ACL*.

Alok Sankar and Sujith Ravi. 2021. Locality sensitive hashing for dynamic text embeddings. In *International Conference on Machine Learning (ICML)*.

Chinnadhurai Sankar, Sujith Ravi, and Zornitsa Kozareva. 2021. Proformer: Towards on-device lsh projection based transformers. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2823–2828.

Tito Svenstrup, Jonas Hansen, and Ole Winther. 2017. Hash embeddings for efficient word representations. *NIPS*.

Wikipedia contributors. 2023. Rolling hash — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Rolling_hash&oldid=1168768744. [Online; accessed 23-October-2023].