https://doi.org/10.53656/math2024-3-1-lex

Science in Education
Научно-методически статии

LEXICAL REPRESENTATION OF DENSE NUMERICAL VECTORS: INTRODUCING LANGVEC

Simeon Emanuilov, Aleksandar Dimov
Sofia University "St. Kliment Ohridski" (Bulgaria)

Abstract. High-dimensional numerical vectors are widely used in machine learning for searching and indexing data. However, it is often difficult for users to interpret their meaning. To address this, we introduce a novel approach that transforms dense vectors into human-readable lexical representations using a percentile-based mapping approach. The essence of the approach is a mapping of words from a predefined/custom lexicon to vectors based on their relative local magnitudes. This way, it enables intuitive visualization of the semantic similarities and differences between complex data points and allows for domain-specific interpretability. It provides an easy way to deduplicate dense vectors (even near-duplicates) and can generate locality-aware hash-like representations, which can be used for efficient indexing and retrieval in various applications. The approach has also been implemented in an open-source library called LangVec. The paper provides examples on LangVec usage and highlights the key applications, including semantic search, recommendation systems, and clustering of numerical data into a human-readable format.

Keywords: interpretable machine learning; vector representations; lexical mapping; semantic similarity; clustering; recommendation systems

1. Introduction

High-dimensional numerical vectors have become the standard representation for encoding information in various machine learning and artificial intelligence applications (Li et al. 2023). In many systems, such as semantic search and recommendation engines, dense vector representations are employed for efficient and accurate information retrieval (Li et al. 2023; Johnson et al. 2019). Dense vectors are compact, fixed-length numerical representations that capture the essential features and relationships of the input data. However, the high dimensionality of these vectors and their numeric nature add complexity when it comes to interpretation. Creators of these systems need to understand how the vectors contribute to the results and trust and rely on

the system's recommendations (Walmsley 2021). Existing techniques for vector interpretation, such as dimensionality reduction and visualization (Jolliffe 2002; Van der Maaten & Hinton 2008; McInnes et al. 2018), often fall short in providing a direct and intuitive interpretation of individual vectors.

To address this gap, here we introduce a novel approach that enables system creators to understand the properties of high-dimensional numerical embeddings in a more intuitive and accessible manner. It is based on incorporating practices from the explainable AI field, such as dimensionality reduction and locality-aware hashing (He & Niyogi 2023; Ahmadian et al. 2022; Mohseni et al. 2021). By mapping vectors to human-readable lexical representations, this method bridges the divide between high-dimensional dense vectors and human understanding.

The approach is also implemented into by an open-source programming library, called LangVec[1], which is written in the Python programming language. LangVec can be particularly beneficial in semantic search systems, enabling hash-like representations with locality awareness and finding exact and near-duplicates and approximate nearest neighbors (ANN) operations.

The paper is structured as follows: Section 2 makes an overview of the related work; Section 3 presents the approach implemented in LangVec; Section 4 provides some usage examples; Section 5 makes a short discussion about pros and cons of the approach and finally, Section 6 makes the concluding remarks of the paper.

2. Related Work

The problem of interpreting and communicating high-dimensional vector representations has been explored from various perspectives in the machine learning and natural language processing communities. Several research areas are directly related to our work, including the use of local descriptors and locality-sensitive hashing (LSH) for efficient indexing and retrieval (2.1), as well as dimensionality reduction techniques (2.2). More generally, several broader research directions are directly related to ours, including word embeddings and language models (2.3), concept activation vectors and probing (2.4), and explainable AI and interpretable machine learning (2.5).

In the following subsections, we discuss these related research areas in more detail, examining their contributions and limitations, and highlighting how our approach complements and extends existing techniques.

2.1. Local descriptors and locality-sensitive hashing (LSH)

The work (Ke et al. 2004) focuses on using local image descriptors (so called PCA-SIFT) for near-duplicate image detection and sub-image retrieval, while LangVec works with general high-dimensional vectors and maps them

to interpretable lexical representations. Their approach is tailored for the image domain, whereas LangVec is a more general-purpose tool for high-dimensional embeddings. Additionally, LangVec provides a way to construct compact, meaningful hash-like representations that preserve locality, which can be useful for indexing and retrieval in various domains beyond images.

LSH, that was proposed by (Indyk & Motwani 1998) and further developed by (Gionis et al. 1999), is an approximate similarity search technique that works efficiently even for high-dimensional data. LangVec leverages LSH for efficient indexing and retrieval of high-dimensional vectors but extends its functionality by mapping the embeddings to readable lexical representations.

2.2. Dimensionality reduction

Dimensionality reduction techniques like Principal Component Analysis (PCA, Jolliffe 2002), t-SNE (Van der Maaten & Hinton 2008), and UMAP (McInnes et al. 2018) aim to project high-dimensional data into lower-dimensional spaces while preserving important structures and relationships. While useful for visualizing patterns and clusters, these methods often lack direct interpretability compared to lexical representations. Recent surveys (Sorzano et al. 2014; Reddy et al. 2020) highlight the growing interest in techniques that can handle non-linear relationships and adapt to local data structures, such as manifold learning (ISOMAP, Locally Linear Embedding, Laplacian Eigenmaps). PCA remains widely used due to its simplicity and interpretability, with variants like robust PCA, sparse PCA, and kernel PCA addressing specific challenges across various domains (Reddy et al. 2020).

The choice of dimensionality reduction technique significantly impacts machine learning performance, depending on dataset size and complexity (Reddy et al. 2020). Locality Preserving Projections (LPP), described in (He & Niyogi 2003), is a classical linear method that preserves local structure by incorporating neighborhood information into a graph and computing a linear transformation matrix. However, LPP's reliance on the original feature space, which may contain noise and irrelevant features, can degrade performance. To address this, Wang et al. (Wang et al. 2020) proposed Locality Adaptive Preserving Projections (LAPP), which adaptively determines neighbors and relationships in the optimal subspace, improving robustness.

While both LPP and LangVec aim to preserve locality, LPP is a linear projection technique, whereas LangVec uses a non-linear mapping based on percentile binning and a predefined lexicon, offering a novel approach to interpretable dimensionality reduction.

2.3. Word embeddings and language models

Traditional word embedding models like Word2Vec (Mikolov et al. 2013) and GloVe (Pennington et al. 2014) learn dense vector representations that

capture semantic relationships but lack interpretability in individual dimensions. More recent pre-trained language models (PLMs) like BERT (Devlin et al. 2018) and GPT (Radford et al. 2019) generate contextualized word embeddings that are even more complex and challenging to interpret directly (Mars 2022).

Techniques such as attention visualization (Clark et al. 2019), probing tasks (Conneau et al. 2018), and layer-wise relevance propagation (Voita et al. 2019) have been proposed to address the interpretability challenge in PLMs, but they often provide only partial or indirect insights into the model's internal reasoning process. Recent work on distilling knowledge from large language models into compact and versatile text embeddings, such as Gecko (Lee et al. 2024; Jiao et al. 2019; Sanh et al. 2019), shows a trend towards smaller embedding dimensions and model sizes while maintaining high performance, aligning with the goal of LangVec.

Although LangVec does not directly work with word embeddings or language models, it shares the objective of making high-dimensional vector representations more interpretable and accessible by mapping dense vectors to human-readable lexical representations.

2.4. Concept activation vectors and probing

Concept Activation Vectors (CAVs) (Kim et al. 2018) are a technique for interpreting neural network internal representations in terms of human-readable concepts. This approach has been applied to various domains, such as computer vision (Ghorbani et al. 2019) and natural language processing (Wei et al. 2021), to gain insights into the learned representations and their alignment with human-defined concepts.

Similarly, probing tasks (Conneau et al. 2018) are designed to analyze the linguistic knowledge encoded in neural language models by training classifiers to predict specific properties from the model's hidden states. These tasks cover various linguistic phenomena, such as part-of-speech tagging, dependency parsing, and coreference resolution (Tenney et al. 2019).

While these approaches provide valuable insights into the presence of specific concepts or properties in a model, they do not directly map vectors to lexical representations. The learned classifiers in CAVs and probing tasks operate on the model's internal activation space, often high-dimensional and not directly interpretable by humans.

2.5. Explainable AI and interpretable machine learning

The field of Explainable AI (XAI, Gunning & Aha 2019) aims to develop methods for understanding and interpreting the decisions made by machine learning models. Techniques such as LIME (Ribeiro et al. 2016), SHAP (Lundberg & Lee 2017), and Grad-CAM (Selvaraju et al. 2017) provide

explanations for individual predictions by identifying influential input features or highlighting relevant regions in an image. However, these explanations are typically specific to a single example and do not provide a global mapping between vector representations and readable concepts.

Existing work in XAI has focused on developing more general and model-agnostic explanation methods, such as TCAV (Kim et al. 2018), which provides global explanations of a model's behavior in terms of human-defined concepts and counterfactual explanations (Wachter et al. 2017; Verma et al. 2020), which identify minimal changes needed in the input to alter the model's prediction.

Mapping high-dimensional embeddings to lexical representations offers a complementary approach to existing interpretability techniques, as most XAI techniques still operate on the input feature space or the model's internal representations, which may not always align with human-interpretable concepts. The lexical representations can be used with dimensionality reduction, probing tasks, and explainable AI methods to enhance further the interpretability and communicability of vector-based models (Ahmadian et al. 2022; Mohseni et al. 2021).

LangVec offers a unique combination of interpretability, locality preservation, and compact hash-like representations that can be valuable in various applications, such as semantic search, recommendation systems, and data deduplication.

3. Approach

The essence of the approach, which is further implemented in LangVec, is to map high-dimensional numerical vectors to human-readable lexical representations. This is achieved through a percentile-based mapping of vector magnitudes to words from a predefined lexicon (Li et al. 2023). The process can be divided into three key steps: (1) lexicon definition, (2) percentile calculation, and (3) vector-to-word mapping, as illustrated in *Figure 1* and further explained below.

3.1. Lexicon definition

The first step is to define a lexicon of words or phrases that will represent the embeddings. By default, LangVec uses a predefined lexicon of common English short words, but users can provide their custom lexicons. The choice of lexicon allows for domain-specific interpretability and can be tailored to the specific data and use case.

The lexicon distribution D is calculated as $D = (d_1, d_2, \ldots, d_{L-1})$, where L is the size of the lexicon and $d_i = i * \frac{100}{L-1}$ for $i = 1, 2, \ldots, L-1$.

This equation shows how the $L - 1$ percentiles for mapping vector magnitudes to words are evenly distributed across the range $[0, 100]$. The lexicon distribution is calculated using equally spaced percentiles to ensure a balanced mapping of vector magnitudes to words across the entire range $[0, 100]$. This approach allows for a more uniform representation of the vector space and prevents skewed mappings that may arise from uneven word assignments.

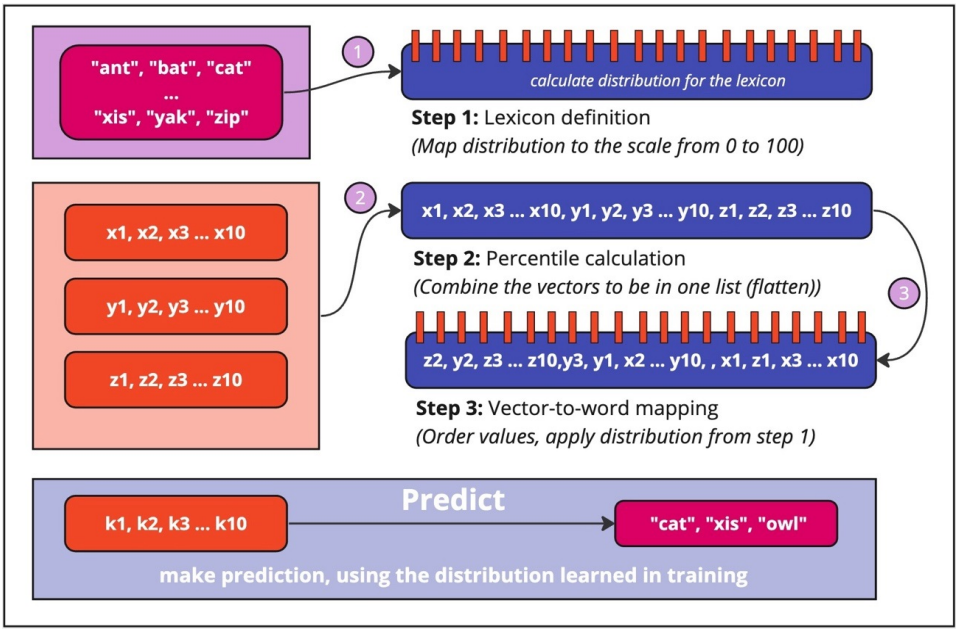

Figure 1. Illustration of LangVec workflow, from input vectors to lexical representations

3.2. Percentile calculation

To map vectors to words, we need to calculate percentiles based on the distribution of vector magnitudes in the training data. The fit function takes a list of numerical vectors and calculates percentiles corresponding to each word in the lexicon. By default, these percentiles are evenly distributed across the range of magnitudes, but this can be adjusted to accommodate different distribution shapes.

Calculating percentiles: Given a set of N numerical vectors $V = \{v_1, v_2, \ldots, v_N\}$, where each vector v_j has M dimensions, the percentiles P are calculated as $P = (p_1, p_2, \ldots, p_{L-1})$, where $p_i = percentile(flatten(V), d_i)$ for $i = 1, 2, \ldots, L - 1$.

The $flatten(V)$ function takes the set of N numerical vectors V and concatenates all the elements of these vectors into a single one-dimensional array. This is done to simplify the percentile calculation process by treating the elements of all vectors as a single dataset. For example, if V consists of three vectors [0.1, 0.2, 0.3], [0.4, 0.5, 0.6], and [0.7, 0.8, 0.9], then $flatten(V)$ would return the array

$$[0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9].$$

The $percentile(X,\ d)$ function calculates the d^{th} percentile of the data in the one-dimensional array X. The d^{th} percentile is the value below which d percent of the data falls. For example, if $d = 50$, the function will return the median value of the data in X. If $d = 25$, it will return the first quartile ($Q1$), and if $d = 75$, it will return the third quartile ($Q3$). The percentile function is used to determine the threshold values for mapping the vector magnitudes to words in the lexicon.

3.3. Vector-to-word mapping

Once the percentiles are calculated, LangVec can map new vectors to their corresponding lexical representations. The predict function takes a single vector and assigns each chunk of the vector (default chunk size is 3) to a word from the lexicon based on its mean magnitude relative to the calculated percentiles.

Given an input vector v with M dimensions, a lexicon W of size L, and a set of percentiles $P = (p_1, p_2, \ldots, p_{L-1})$, LangVec maps v to a sequence of words $S = (s_1,\ s_2, \ldots, s_K)$ using the following steps:

- Divide the input vector v into K chunks, each containing C elements, where C is the chunk size $K = \lfloor M\ /\ C \rfloor$.

This step ensures that the input vector is divided into equal-sized chunks, with the last chunk potentially having fewer elements if the vector length is not divisible by the chunk size.

- For each chunk c_j, where $j = 1, 2, \ldots, K$, calculate the mean magnitude $m_j = \text{mean}(c_j)$.
- Compare each mean magnitude m_j to the percentiles in P to determine the corresponding word $s_j = W[b_j]$, where $(b_j = \text{sum}(m_j > p_i \text{ for } i = 1, 2, \ldots, L-1)$ respectively, b_j represents the number of percentiles that m_j exceeds, and $W[b_j]$ retrieves the word at index b_j in the lexicon W.

3.4. Sensitivity to input changes

The percentile-based mapping approach allows for a certain level of robustness to small changes in the input vector. When the input vector is slightly modified, the resulting lexical representation may remain unchanged or exhibit only local changes, depending on the magnitude of the modification and the distribution of the training data.

For example, consider an input vector [0.2, 0.5, 0.8, 0.1, 0.3, 0.9] mapped to the lexical representation [$'high'$, $'medium'$] based on the percentiles learned from the training data. If we slightly modify the last few elements of the vector to [0.2, 0.5, 0.8, **0.2**, **0.31**, **0.91**], the resulting lexical representation may still be [$'high'$, $'medium'$], as the change is small enough not to cross the percentile boundaries. This tool's property can be advantageous in specific applications, such as similarity-based retrieval, deduplication, or clustering, where slight variations in the input should not significantly alter the output.

4. Usage

In this section, we provide a step-by-step guide on how to use the LangVec library, along with a code example showcasing its key functionalities.

4.1. Basic usage

Basic usage of LangVec library should include initialization of the main object, fitting the model to a dataset, and predicting lexical representations for new vectors. Listing 1 presents an example of how to use LangVec to map a set of numerical vectors to their lexical representations:

Listing 1: Python code demonstrating LangVec's basic usage

```
import numpy as np
from langvec import LangVec
np.random.seed(42)
# Initialize LangVec
lv = LangVec()
NUM_VECTORS, DIMENSIONS = 1000, 10
# Generate some random data
vectors = [np.random.uniform(0, 1, DIMENSIONS) for _
          in range(NUM_VECTORS)]
# Fit to this data (getting know the distribution)
lv.fit(vectors)
# Example vector for prediction
input_vector = np.random.uniform(0, 1, DIMENSIONS)
print(lv.predict(input_vector))
```

4.2. Customization options

The library provides several customization options, allowing users to tailor the library's behavior to their specific needs. Here are some of the key parameters:

Table 1. Default values of main parameters in LangVec

Parameter	Default value	Description	Scope
lexicon	*constants.LEXICON*	The list of words/characters to use for mapping.	LangVec
chunk_size	3	The number of vector elements to map to each word/character.	LangVec
summarized	False	Whether to summarize the output.	predict
padding	True	Whether to pad the last chunk with zeros.	predict
max_samples	10^7	The maximum number of samples to use for fitting.	fit, update

The default value *constants.LEXICON* in Table 1 is a set of 26 short English words, which can be overridden by the users with a new list of words.

These parameters have different scopes (LangVec is the highest scope, while *predict, fit*, and *update* are a lower scope) and can be set when initializing the LangVec object or when calling the predict method. For example:

Listing 2: Code snippet illustrating how to customize LangVec by specifying a custom lexicon, adjusting the chunk size and summarization options

```
# Initialize LangVec with a custom lexicon and chunk size
custom_lexicon = ["small", "medium", "large"]
lv = LangVec(lexicon=custom_lexicon, chunk_size=4)
# Map a vector and summarize the output
lexical_rep = lv.predict(new_vector, summarized=True)
```

The library accepts input data as NumPy arrays or lists of arrays, making it compatible with popular data processing and modeling tools such as pandas, scikit-learn, and TensorFlow.

4.3. Case study

To demonstrate the practical application and effectiveness of LangVec in a real-world scenario, we conducted a case study[2] using PostgreSQL to benchmark string similarity search using the Levenshtein (Yujian & Bo 2007) distance metric. The goal was to evaluate the performance of finding near-duplicate strings in a large dataset and showcase the key benefits of the proposed approach.

4.4. Configuration

We set up a PostgreSQL database on a server with the following characteristics: Intel(R) Xeon(R) E-2274G CPU @ 4.00GHz, 64 GB DDR4 RAM, and 512 GB NVMe SSD. Then we populated it with 10 million strings of length 32, consisting of lowercase letters from 'a' to 'z'. The strings were stored in a table with an index created on the *str* column using the pg_trgm extension for efficient trigram-based string similarity search.

We defined a function $find\_similar\_strings$ that takes an input string and a maximum Levenshtein distance as parameters and returns a table of up to 100 strings from the strings table that are within the specified distance from the input string, sorted by distance.

To benchmark the performance, we generated random input strings and executed the $find\_similar\_strings$ function with different maximum Levenshtein distances (5, 10, and 20). We measured the execution time for each query to assess the efficiency of the string similarity search.

4.5. Results

The benchmark results showed that the execution times for finding similar strings using LangVec in PostgreSQL were consistent across different maximum Levenshtein distances, ranging from 4.9 to 5 seconds for a dataset of 10 million strings.

The query plan involved a function scan on $find\_similar\_strings$ followed by sorting the results based on the distance.

To further validate the effectiveness of LangVec, we conducted an additional experiment where we picked 10 random vectors from the generated set and performed a search on them. In all 10 cases, the same vector was returned with a distance of 0, which highlights the capability of LangVec to accurately identify exact matches within the dataset.

The case study highlights the practical application of LangVec in a real-world scenario, demonstrating its efficiency, accuracy, and integration capa-

bilities with PostgreSQL. The approach successfully addresses the challenge of finding near-duplicate strings in large datasets, providing a valuable tool for deduplication and similarity search tasks.

4.6. Benchmark

The benchmark tests were performed on the server from Section 4.4. The benchmarking script available in the LangVec repository[3] generates sample data consisting of random vectors with a specified number of dimensions. The script measures the fitting time for different dataset sizes ($10^3, 10^4, 10^5$, and 10^6 256 dimensional vectors) and the average prediction time for 10,000 random embeddings.

The results, as shown in Table 2, demonstrate the efficiency of LangVec in terms of fitting and prediction times for various dataset sizes. The fitting time increases with the number of embeddings, while the prediction time remains relatively constant. It is important to note that the actual performance may vary depending on the hardware configuration and the characteristics of the input data.

Table 2. Benchmark results showcase LangVec performance on different dataset sizes, ranging from 10^3 to 10^6 256-dimensional vectors. The table presents the fitting time and prediction time for each dataset size.

Number of vectors*	Fitting time (seconds)	Prediction time (seconds)
10^3	0.0130	$4.25 * 10^{-4}$
10^4	0.0925	$4.11 * 10^{-4}$
10^5	0.8658	$4.53 * 10^{-4}$
10^6	9.9734	$4.13 * 10^{-4}$

For extremely large datasets that exceed the available machine memory, users can implement random sampling of the input data to train the model. By selecting a representative subset of the data, users can effectively capture the underlying distribution while reducing the computational burden.

4.7. Error handling

The library includes error handling to provide informative messages when issues arise during the execution of the library's functions. Error handling helps users identify and correct problems with their input data. Some common errors include:

- Attempting to call the prediction method before fitting the model.
- Passing invalid input data to the fit or predict methods.

• Inconsistencies between the lexicon size and the learned percentiles during the prediction phase.

By providing clear and informative error messages, our implementation helps users quickly identify and resolve issues, improving the overall user experience and reducing the debugging time. The error handling also ensures that the library is used correctly and that the input data meets the expected format, contributing to the reliability and robustness of the system.

5. Discussion

While LangVec offers a novel approach to mapping high-dimensional vectors to human-readable lexical representations, it has some limitations. Firstly, LangVec relies on the training data distribution to learn the mapping, and if the data distribution changes significantly between the training and testing phases, the learned mapping may not generalize well, leading to suboptimal results. This highlights the importance of the fit method, which allows the library to adapt to the specific distribution of the dataset at hand. Secondly, the lexical representations generated by LangVec may not always be meaningful, as they are based on a predefined lexicon and may not capture the semantic content of the data, although readability should still be improved.

Moreover, the locality-preserving property may not be suitable for all applications, as in some cases, global structure may be more important than local structure, and other dimensionality reduction techniques, such as PCA or t-SNE, may be more appropriate. Another aspect to consider is the occurrence of collisions when using LangVec as a hash-like function, where different input vectors map to the same lexical representation. The likelihood of collisions depends on factors such as the chunk size, lexicon length, and embedding dimension, and can either be avoided or leveraged as a desired effect, depending on the specific use case.

For applications like duplication removal and similar content detection, having hashing collisions (i.e., a variation-tolerant, locality-aware algorithm) is a must-have feature, as LangVec's ability to map similar vectors to the same or similar lexical representations allows for efficient identification of duplicate or near-duplicate items (Ke et al. 2004). On the other hand, for applications where unique mappings are required, such as certain indexing or retrieval tasks, collisions should be minimized through careful selection of the chunk size, lexicon length, and embedding dimension.

Despite these limitations, LangVec offers a unique and valuable approach to mapping high-dimensional vectors to human-readable representations. One of its main applications is in constructing hash-like representations for dense vectors in semantic search systems that use techniques like FAISS[4] clusters via K-Nearest Neighbors (KNN) (Johnson et al. 2019), providing an effective

and easy-to-use alternative to traditional hashing methods (Indyk & Motwani 1998; Gionis et al. 1999). LangVec's locality-aware and change-tolerant property makes it particularly useful in search systems, where minor variations in the input vectors should not dramatically alter the search results.

Furthermore, in the context of model debugging and interpretation, the library can be used to find the internal representations and decision-making stages of complex machine learning models (Montavon 2019). By mapping intermediate embedding values to lexical representations, developers and researchers can gain insights into the distribution and nature of the data, facilitating model debugging, optimization, and explanatory analysis.

6. Conclusion

This paper introduced LangVec, an approach and open-source implementation that maps high-dimensional numerical vectors to human-readable lexical representations. By leveraging a percentile-based mapping approach, our method reduces the distance between machine learning models and human understanding, enabling intuitive interpretation and communication of vector-based data and model outputs.

The main contributions of the proposed technique are:

- A novel approach for mapping dense numerical vectors to interpretable lexical representations or hash-like strings, using fitting methods to learn underlying data distribution.
- An implementation, benchmark, and case study of this approach available for use by the community.

We highlighted several main applications of LangVec, including semantic search and recommendation systems, deduplication, data exploration and clustering, model interpretation and debugging, and human-in-the-loop learning. These examples demonstrate the potential for LangVec to enhance interpretability and help build various types of applications using embeddings.

We also discussed the utility of the library in constructing meaningful hash-like strings for semantic search systems (Johnson et al. 2019; Ahmadian et al. 2022) and its role as a location-aware dimensionality reduction technique, particularly useful in hybrid vector representations (Turner et al. 2021), but also when indexes are reshuffled in indexing strategies like using FAISS.

LangVec should serve as a tool for researchers and practitioners seeking to make their model outputs more accessible and interpretable to a broader audience. By providing a human-friendly API interface to complex numerical representations, LangVec facilitates better understanding, trust, and collaboration between technical and non-technical stakeholders (Mohseni et al. 2021).

Future work on LangVec could explore several directions, such as:

• Incorporating more advanced natural language processing techniques to generate more fluent and contextually relevant lexical representations instead of static lexicon.

• Developing interactive visualization tools to enable intuitive exploration of high-dimensional datasets and model outputs.

Finally, we invite the machine learning community to explore, contribute to, and provide feedback on LangVec, as we strive to make embedding representations more interpretable and accessible to a broader range of users.

Acknowledgements

This study is financed by the European Union-NextGenerationEU, through the National Recovery and Resilience Plan of the Republic of Bulgaria, project № BG-RRP-2.004-0008-C01

NOTES

1. Language of Vectors (LangVec) implementation. https://github.com/s-emanuilov/langvec
2. Near duplication detection with PostgreSQL and LangVec approach. https://github.com/s-emanuilov/LangVec/blob/main/docs/NEAR_DUPLICATION_BENCHMARK.md
3. LangVec benchmark script. https://github.com/s-emanuilov/LangVec/blob/main/benchmark.py
4. FAISS, Meta Platforms, Inc. https://github.com/facebookresearch/faiss

REFERENCES

AHMADIAN, M., AHMADI, M., AHMADIAN, S., 2022. A Reliable Deep Representation Learning to Improve Trust-aware Recommendation Systems. *Expert Systems with Applications*, vol. 197, pp.116697. doi: 10.1016/j.eswa.2022.116697

CLARK, K., KHANDELWAL, U., LEVY, O., MANNING, C.D., 2019. *What Does BERT Look At? An Analysis of BERTS's Attention.* arXiv:1906.04341

CONNEAU, A., KRUSZEWSKI, G., LAMPLE, G., BARRAULT, L., BARONI, M., 2018. *What you can cram into a single vector: Probing sentence embeddings for linguistic properties.* arXiv:1805.01070

DEVLIN, J., CHANG, M.W., LEE, K.,TOUTANOVA, K., 2018. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* arXiv:1810.04805

GHORBANI, A., WEXLER, J., ZOU, J.Y., KIM, B., 2019. Towards Automatic Concept-based Explanations. *Advances in Neural Information Processing Systems 32 (NeurIPS 2019).* arXiv:1902.03129

GIONIS, A., INDYK, P., MOTWANI, R., 1999. Similarity search in high dimensions via hashing. In *Vldb*, vol. 99, no. 6, pp. 518 – 529. https://www.vldb.org/conf/1999/P49.pdf

GUNNING, D., AHA, D., 2019. DARPA's explainable artificial intelligence (XAI) program. *AI magazine*, vol. 40, no. 2, pp. 44 – 58.

HE, X., NIYOGI, P., 2003. Locality preserving projections. *Advances in Neural Information Processing Systems 16 (NIPS 2003).*

INDYK, P., MOTWANI, R., 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. *Proceedings of thirtieth annual ACM symposium on Theory of computing*, pp. 604 – 613. doi: 10.1145/276698.276876

JIAO, X., YIN, Y., SHANG, L., JIANG, X., CHEN, X., LI, L., WANG, F., LIU, Q., 2019. *TinyBERT: Distilling BERT for Natural Language Understanding.* arXiv:1909.10351.

JOHNSON, J., DOUZE, M., JÉGOU, H., 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535 – 547.

JOLLIFFE, I.T., 2002. *Principal component analysis for special types of data*, Springer, New York, pp. 338 – 372. ISBN 978-0-387-22440-4

KE, Y., SUKTHANKAR, R., HUSTON, L., 2004. Efficient near-duplicate detection and sub-image retrieval. In *ACM multimedia*, vol. 4, no. 1, pp. 5.

KIM, B., WATTENBERG, M., GILMER, J., CAI, C., WEXLER, J., VIEGAS, F., 2018. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In *International conference on machine learning*, pp. 4186 – 4195. arXiv:1711.11279

LEE, J., DAI, Z., REN, X., CHEN, B., CER, D., COLE, J.R., HUI, K., BORATKO, M., KAPADIA, R., DING, W., LUAN, Y., DUDDU, S.M.K., ABREGO, G.H., SHI, W., GUPTA, N., KUSUPATI, A., JAIN, P., JONNALAGADDA, S.R., CHANG, M-W., NAIM, I., 2024. *Gecko: Versatile Text Embeddings Distilled from Large Language Models.* arXiv:2403.20327.

LI, H., WANG, J., ZHENG, Y., WANG, L., ZHANG, W., SHEN, H.W., 2023. Compressing and interpreting word embeddings with latent space regularization and interactive semantics probing. *Information Visualization*, vol. 22, no. 1, pp. 52 – 68. arXiv:2403.16815

LUNDBERG, S.M., LEE, S.I., 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems 30 (NIPS 2017)*.

MARS, M., 2022. From word embeddings to pre-trained language models: A state-of-the-art walkthrough. *Applied Sciences*, vol. 12, no. 17, p. 8805.

MCINNES, L., HEALY, J., MELVILLE, J., 2018. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.* arXiv:1802.03426

MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G.S., DEAN, J., 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems 26 (NIPS 2013)*.

MOHSENI, S., ZAREI, N., RAGAN, E.D., 2021. A multidisciplinary survey and framework for design and evaluation of explainable AI systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 11, no. 3 – 4, pp. 1 – 45.

MONTAVON, G., BINDER, A., LAPUSCHKIN, S., SAMEK, W., MÜLLER, K.R., 2019. *Explainable AI: interpreting, explaining and visualizing deep learning.* Springer, LNCS, 11700.

PENNINGTON, J., SOCHER, R., MANNING, C.D., 2014. Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532 – 1543.

RADFORD, A., WU, J., CHILD, R., LUAN, D., AMODEI, D., SUTSKEVER, I., 2019. Language models are unsupervised multitask learners. *OpenAI blog*, vol. 1, no. 8, p. 9.

REDDY, G.T., REDDY, M.P.K., LAKSHMANNA, K., KALURI, R., RAJPUT, D.S., SRIVASTAVA, G., BAKER, T., 2020. Analysis of dimensionality reduction techniques on big data. *IEEE Access*, vol. 8, pp. 54776 – 54788.

RIBEIRO, M.T., SINGH, S., GUESTRIN, C., 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135 – 1144. arXiv:1602.04938

SANH, V., DEBUT, L., CHAUMOND, J., WOLF, T., 2019. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* arXiv:1910.01108

SELVARAJU, R.R., COGSWELL, M., DAS, A., VEDANTAM, R., PARIKH, D., BATRA, D., 2017. Grad-cam: Visual explanations from

deep networks via gradient-based localization. *Proceedings of the IEEE international conference on computer vision*, pp. 618 – 626.

SORZANO, C.O.S., VARGAS, J., MONTANO, A.P., 2014. *A survey of dimensionality reduction techniques.* arXiv:1403.2877

TENNEY, I., DAS, D., PAVLICK, E., 2019. *BERT rediscovers the classical NLP pipeline.* arXiv:1905.05950

TURNER, C.J., MA, R., CHEN, J., OYEKAN, J., 2021. Human in the Loop: Industry 4.0 technologies and scenarios for worker mediation of automated manufacturing. *IEEE access*, vol. 9, pp. 103950 – 103966.

VAN DER MAATEN, L., HINTON, G., 2008. Visualizing data using t-SNE. *Journal of machine learning research*, vol. 9, pp. 2579 – 2605.

VERMA, S., BOONSANONG, V., HOANG, M., HINES, K.E., DICKERSON, J.P., SHAH, C., 2020. *Counterfactual explanations and algorithmic recourses for machine learning: A review.* arXiv:2010.10596

VOITA, E., TALBOT, D., MOISEEV, F., SENNRICH, R., TITOV, I., 2019. *Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned.* arXiv:1905.09418

WACHTER, S., MITTELSTADT, B., RUSSELL, C., 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law & Technology*, vol. 31, no. 2, pp. 841 – 847.

WALMSLEY, J., 2021. Artificial intelligence and the value of transparency. *AI & society*, vol. 36, no. 2, pp. 585 – 595.

WANG, A., ZHAO, S., LIU, J., YANG, J., LIU, L., CHEN, G., 2020. Locality adaptive preserving projections for linear dimensionality reduction. *Expert Systems with Applications*, vol. 151, p. 113352.

WEI, X., GALES, M.J., KNILL, K.M., 2021. Analysing bias in spoken language assessment using concept activation vectors. *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7753 – 7757.

YUJIAN, L., BO, L., 2007. A normalized Levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091 – 1095.

✉ **Simeon Emanuilov**
ORCID iD: 0000-0002-2295-4513
Faculty of Mathematics and Informatics
Sofia University "St. Kliment Ohridski"
8, James Bourchier Blvd.
1164 Sofia, Bulgaria
E-mail: ssemanuilo@fmi.uni-sofia.bg

✉ **Dr. Aleksandar Dimov, Prof.**
ORCID iD: 0000-0001-6197-1212
Faculty of Mathematics and Informatics
Sofia University "St. Kliment Ohridski"
8, James Bourchier Blvd.
1164 Sofia, Bulgaria
E-mail: aldi@fmi.uni-sofia.bg