Imputing out-of-vocabulary embeddings with LOVE makes language models robust with little cost
Lihu Chen, Gaël Varoquaux, Fabian Suchanek

To cite this version:
Lihu Chen, Gaël Varoquaux, Fabian Suchanek. Imputing out-of-vocabulary embeddings with LOVE makes language models robust with little cost. ACL 2022 - 60th Annual Meeting of the Association for Computational Linguistics, May 2022, Dublin, Ireland. hal-03613101

HAL Id: hal-03613101
https://hal.archives-ouvertes.fr/hal-03613101
Submitted on 19 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Imputing Out-of-Vocabulary Embeddings with LOVE Makes Language Models Robust with Little Cost

Lihu Chen\(^1\), Gaël Varoquaux\(^2\), Fabian M. Suchanek\(^1\)

\(^1\) LTCI & Télécom Paris & Institut Polytechnique de Paris, France
\(^2\) Soda, Inria Saclay & CEA & Université Paris-Saclay, France

{lihu.chen, fabian.suchanek}@telecom-paris.fr
{gael.varoquaux}@inria.fr

Abstract

State-of-the-art NLP systems represent inputs with word embeddings, but these are brittle when faced with Out-of-Vocabulary (OOV) words. To address this issue, we follow the principle of mimick-like models to generate vectors for unseen words, by learning the behavior of pre-trained embeddings using only the surface form of words. We present a simple contrastive learning framework, LOVE, which extends the word representation of an existing pre-trained language model (such as BERT), and makes it robust to OOV with few additional parameters. Extensive evaluations demonstrate that our lightweight model achieves similar or even better performances than prior competitors, both on original datasets and on corrupted variants. Moreover, it can be used in a plug-and-play fashion with FastText and BERT, where it significantly improves their robustness.

1 Introduction

Word embeddings represent words as vectors (Mikolov et al., 2013a,b; Pennington et al., 2014). They have been instrumental in neural network approaches that brought impressive performance gains to many natural language processing (NLP) tasks. These approaches use a fixed-size vocabulary. Thus they can deal only with words that have been seen during training. While this works well on many benchmark datasets, real-word corpora are typically much noisier and contain Out-of-Vocabulary (OOV) words, i.e., rare words, domain-specific words, slang words, and words with typos, which have not been seen during training. Model performance deteriorates a lot with unseen words, and minor character perturbations can flip the prediction of a model (Liang et al., 2018; Belinkov and Bisk, 2018; Sun et al., 2020; Jin et al., 2020). Simple experiments (Figure 1) show that the addition of typos to datasets degrades the performance for text classification models considerably.

To alleviate this problem, pioneering work pretrained word embeddings with morphological features (sub-word tokens) on large-scale datasets (Wieting et al., 2016; Bojanowski et al., 2017; Heinzlinger and Strube, 2017; Zhang et al., 2019). One of the most prominent works in this direction is FastText (Bojanowski et al., 2017), which incorporates character n-grams into the skip-gram model. With FastText, vectors of unseen words can be imputed by summing up the n-gram vectors. However, these subword-level models come with great costs: the requirements of pre-training from scratch and high memory footprint. Hence, several simpler approaches have been developed, e.g., MIMICK (Pinter et al., 2017), BoS (Zhao et al., 2018) and KVQ-FH (Sasaki et al., 2019). These use only the surface form of words to generate vectors for unseen words, through learning from pre-trained embeddings.

Although MIMICK-like models can efficiently reduce the parameters of pre-trained representa-
tions and alleviate the OOV problem, two main challenges remain. First, the models remain bound in the trade-off between complexity and performance: The original MIMICK is lightweight but does not produce high-quality word vectors consistently. BoS and KVQ-FH obtain better word representations but need more parameters. Second, these models cannot be used with existing pre-trained language models such as BERT. It is these models, however, to which we owe so much progress in the domain (Peters et al., 2018; Devlin et al., 2019; Yang et al., 2019; Liu et al., 2020). To date, these high-performant models are still fragile when dealing with rare words (Schick and Schütze, 2020), misspellings (Sun et al., 2020) and domain-specific words (El Boukkouri et al., 2020).

We address these two challenges head-on: we design a new contrastive learning framework to learn the behavior of pre-trained embeddings, dubbed LOVE, Learning Out-of-Vocabulary Embeddings. Our model builds upon a memory-saving mixed input of character and subwords instead of n-gram characters. It encodes this input by a lightweight Positional Attention Module. During training, LOVE uses novel types of data augmentation and hard negative generation. The model is then able to produce high-quality word representations that are robust to character perturbations, while consuming only a fraction of the cost of existing models. For instance, LOVE with 6.5M parameters can obtain similar representations as the original FastText model with more than 900M parameters. What is more, our model can be used in a plug-and-play fashion to robustify existing language models. We find that using LOVE to produce vectors for unseen words improves the performance of FastText and BERT by around 1.4-6.8 percentage points on noisy text – without hampering their original capabilities (As shown in Figure 2).

In the following, Section 2 discusses related work, Section 3 introduces preliminaries, Section 4 presents our approach, Section 5 shows our experiments, and Section 6 concludes. The appendix contains additional experiments and analyses. Our code and data is available at https://github.com/tigerchen52/LOVE

2 Related Work

2.1 Character-level Embeddings

To address OOV problems, some approaches inject character-level features into word embeddings during the pre-training (Wieting et al., 2016; Cao and Rei, 2016; Bojanowski et al., 2017; Heinzerling and Strube, 2017; Kim et al., 2018; Li et al., 2018; Üstün et al., 2018; Piktus et al., 2019; Zhu et al., 2019; Zhang et al., 2019; Hu et al., 2019). One drawback of these methods is that they need to pre-train on a large-scale corpus from scratch. Therefore, simpler models have been developed, which directly mimic the well-trained word embeddings to impute vectors for OOV words. Some of these methods use only the surface form of words to generate embeddings for unseen words (Pinter et al., 2017; Zhao et al., 2018; Sasaki et al., 2019; Fukuda et al., 2020; Jinman et al., 2020), while others use both surface and contextual information to create OOV vectors (Schick and Schütze, 2019a,b). In both cases, the models need an excessive number of parameters. FastText, e.g., uses ~2 million n-gram characters to impute vectors for unseen words.

2.2 Pre-trained Language Models

Currently, the state-of-the-art word representations are pre-trained language models, such as ELMo (Peters et al., 2018), BERT (Devlin et al., 2019) and XLNet (Yang et al., 2019), which adopt subwords to avoid OOV problems. However, BERT is brittle when faced with rare words (Schick and Schütze, 2020) and misspellings (Sun et al., 2020). To make BERT more robust, CharacterBERT (El Boukkouri et al., 2020) and Char-BERT (Ma et al., 2020) infuse character-level features into BERT and pre-train the variant from
The origin of contrastive learning can be traced back to the work by Becker and Hinton (1992) and Bromley et al. (1993). This method has achieved outstanding success in self-supervised representation learning for images (Oord et al., 2018; Hjelm et al., 2018; He et al., 2020; Chen et al., 2020; Grill et al., 2020). The contrastive learning framework learns representations from unlabeled data by pulling positive pairs together and pushing negative pairs apart. For training, the positive pairs are often obtained by taking two randomly augmented versions of the same sample and treating the other augmented examples within a mini-batch as negative examples (Chen et al., 2017, 2020). The most widely used loss is the infoNCE loss (or contrastive loss) (Hjelm et al., 2018; Loellmann et al., 2019). The contrastive learning methods have achieved significant success for image representation (Oord et al., 2018; Logeswaran and Lee, 2018; Chen et al., 2020; He et al., 2020; Chen et al., 2020). The core idea of these methods is to encourage learned representations for positive pairs to be close, while pushing representations from sampled negative pairs apart. The widely used contrastive loss (Hjelm et al., 2018; Logeswaran and Lee, 2018; Chen et al., 2020; He et al., 2020) is defined as:

\[
\ell_{cl} = -\log \frac{e^{\text{sim}(u_i^+ u_i^+)/\tau}}{e^{\text{sim}(u_i^+ u_i^+)/\tau} + \sum e^{\text{sim}(u_i^+ u^-)/\tau}}
\]

### 3 Preliminaries

#### 3.1 Mimick-like Model

Given pre-trained word embeddings, and given an OOV word, the core idea of MIMICK (Pinter et al., 2017) is to impute an embedding for the OOV word using the surface form of the word, so as to mimic the behavior of the known embeddings. BoS (Zhao et al., 2018), KVQ-FH (Sasaki et al., 2019), Robust Backed-off Estimation (Fukuda et al., 2020), and PBoS (Jinman et al., 2020) work similarly, and we refer to them as mimick-like models.

Formally, we have a fixed-size vocabulary set \( \mathcal{V} \), with an embedding matrix \( W \in \mathbb{R}^{|\mathcal{V}| \times m} \), in which each row is a word vector \( u_w \in \mathbb{R}^m \) for the word \( w \). A mimick-like model aims to impute a vector \( v_w \) for an arbitrary word \( w \not\in \mathcal{V} \). The training objective of mimick-like models is to minimize the expected distance between \( u_w \) and \( v_w \):

\[
L_{\text{dis}} = \frac{1}{|\mathcal{V}|} \sum_{w \in \mathcal{V}} \psi(u_w, v_w)
\]

Here, \( \psi(\cdot) \) is a distance function, e.g., the Euclidean distance \( \psi = \|u_w - v_w\|_2^{2} \) or the cosine distance \( \psi = 1 - \cos(u_w, v_w) \). The vector \( v_w \) is generated by the following equation:

\[
v_w = \phi(\zeta(w)), \text{ for } w \in \mathcal{V} \text{ or } w \not\in \mathcal{V}
\]

Here, \( \zeta(\cdot) \) is a function that maps \( w \) to a list of subunits based on the surface form of the word (e.g., a character or subword sequence). After that, the sequence is fed into the function \( \phi(\cdot) \) to produce vectors, and the inside structure can be CNNs, RNNs, or a simple summation function. After training, the model can impute vectors for arbitrary words.

### 3.2 Contrastive Learning

Contrastive learning methods have achieved significant success for image representation (Oord et al., 2018; Chen et al., 2020). The core idea of these methods is to encourage learned representations for positive pairs to be close, while pushing representations from sampled negative pairs apart. The widely used contrastive loss (Hjelm et al., 2018; Logeswaran and Lee, 2018; Chen et al., 2020; He et al., 2020) is defined as:

\[
\ell_{cl} = -\log \frac{e^{\text{sim}(u_i^+ u_i^+)/\tau}}{e^{\text{sim}(u_i^+ u_i^+)/\tau} + \sum e^{\text{sim}(u_i^+ u^-)/\tau}}
\]

---

<table>
<thead>
<tr>
<th>Mimick-like Model</th>
<th>Input</th>
<th>Encoder</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIMICK (2017)</td>
<td>{s, p, e, l}</td>
<td>RNNs</td>
<td>( L_{\text{dis}} )</td>
</tr>
<tr>
<td>BoS (2018)</td>
<td>n-gram subword {spe, pel, ell}</td>
<td>SUM</td>
<td>( L_{\text{dis}} )</td>
</tr>
<tr>
<td>KVQ-FH (2019)</td>
<td>adapted n-gram subword {spe, pel, ell}</td>
<td>Attention</td>
<td>( L_{\text{dis}} )</td>
</tr>
</tbody>
</table>

Table 1: Details of different mimick-like models, with the word spell as an example.
There are five key ingredients in the framework: choice of Hard Negatives, the Loss Function, our Data Augmentation, and the ones in Table 1): the Input Method, the Encoder, that we will detail in the following (similar to the contrastive loss as an objective function (Eq 3). As large as possible. As mentioned before, we use negative pairs (other samples in the same mini-batch) between them while making the distance of negative and positive samples, and we maximize the similarity of positive pairs. An overview of our framework is shown in Figure 3. It is inspired by work in visual representation learning (Chen et al., 2020), but differs in that one of the positive pairs is obtained from pre-trained embeddings instead of using two augmented versions. We adopt five novel types of word-level augmentations and a lightweight Positional Attention Module in this framework. Moreover, we find that adding hard negatives during training can effectively yield better representations. We removed the nonlinear projection head after the encoder layer, because its improvements are specific to the representation quality in the visual field. Furthermore, our approach is not an unsupervised contrastive learning framework, but a supervised learning approach.

Our framework takes a word from the original vocabulary and uses data augmentation to produce a corruption of it. For example, "misspelling" becomes "mispelling" after dropping one letter "s". Next, we obtain a target vector from the pre-trained embeddings for the original word, and we generate a vector for the corrupted word. These two vectors are a pair of positive samples, and we maximize the similarity between them while making the distance of negative pairs (other samples in the same mini-batch) as large as possible. As mentioned before, we use the contrastive loss as an objective function (Eq 3). There are five key ingredients in the framework that we will detail in the following (similar to the ones in Table 1): the Input Method, the Encoder, the Loss Function, our Data Augmentation, and the choice of Hard Negatives.

4 Our Approach: LOVE

LOVE (Learning Out-of-Vocabulary Embeddings) draws on the principles of contrastive learning to maximize the similarity between target and generated vectors, and to push apart negative pairs. An overview of our framework is shown in Figure 3. It is inspired by work in visual representation learning (Chen et al., 2020), but differs in that one of the positive pairs is obtained from pre-trained embeddings instead of using two augmented versions. We adopt five novel types of word-level augmentations and a lightweight Positional Attention Module in this framework. Moreover, we find that adding hard negatives during training can effectively yield better representations. We removed the nonlinear projection head after the encoder layer, because its improvements are specific to the representation quality in the visual field. Furthermore, our approach is not an unsupervised contrastive learning framework, but a supervised learning approach.

Our goal is to use the surface form to impute vectors for words. The question is thus how to design the function $\zeta(\cdot)$ mentioned in Section 3.1 to represent each input word. MIMICK (Pinter et al., 2017) straightforwardly uses the character sequence (see Table 1). This, however, loses the information of morphemes, i.e., sequences of characters that together contribute a meaning. Hence, FastText (Bojanowski et al., 2017) adopts character n-grams. Such n-grams, however, are highly redundant. For example, if we use substrings of length 3 to 5 to represent the word misspelling, we obtain a list with 24 n-gram characters – while only the substrings {mis, spell, ing} are the three crucial units to understand the word. Hence, like BERT, we use WordPiece (Wu et al., 2016) with a vocabulary size of around 30000 to obtain meaningful subwords of the input word. For the word misspelling, this yields {miss, #pel, #ling}. However, if we just swap two letters (as by a typo), then the sequence becomes completely different: {mi, #sp, #sell, #ing}. Therefore, we use both the character sequence and subwords (Figure A1).

We shrink our vocabulary by stemming all words and keeping only the base form of each word, and by removing words with numerals. This decreases the size of vocabulary from 30000 to 21 257 without degrading performance too much (Section A.1).

4.2 Encoder

Let us now design the function $\phi(\cdot)$ mentioned in Section 3.1. We are looking for a function that can encode both local features and global features. Local features are character n-grams, which provide robustness against minor variations such as character swaps or omissions. Global features combine local features regardless of their distance. For the word misspelling, a pattern of prefix and suffix missing can be obtained by combining the local information at the beginning and the end of the word. Conventional CNNs, RNNs, and self-attention cannot extract such local and global information at the same time. Therefore, we design a new Positional Attention Module. Suppose we have an aforementioned mixed input sequence and a corresponding embedding matrix $V \in \mathbb{R}^{|V| \times d}$ where $d$ is the dimension of vectors. Then the input can be represented by a list of vectors: $X = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^{n \times d}$ where $n$ is the
Data Augmentation → Encoder

Pre-trained Embeddings

“missspelling”

Maximize Similarity

Figure 3: The framework of LOVE with an example of the word misspelling.

length of the input. To extract local information, we first adopt positional attention to obtain n-gram features, and then feed them into a conventional self-attention layer to combine them in a global way. This can be written as:

\[
\tilde{X} = SA(PA(X)) W^O
\]  (4)

Here, \(SA\) is a standard multi-head self-attention and \(PA\) is a positional attention. \(W^O \in \mathbb{R}^{d_V \times d_O}\) is a trainable parameter matrix, where \(d_V\) are the dimensions of values in \(SA\) and \(PA\), and \(d_O\) is that of \(\tilde{X}\). As for the Positional Attention, we adopt absolute sinusoidal embeddings (Vaswani et al., 2017) to compute positional correlations:

\[
PA(X) = \text{Softmax}\left(\frac{PP^T}{\sqrt{d}}\right)(X W^V)
\]  (5)

Here, \(P \in \mathbb{R}^{n \times d}\) are the position embeddings, and \(W^V \in \mathbb{R}^{d \times d_V}\) are the corresponding parameters. More details about the encoder are in Appendix C.4.

4.3 Loss Function

In this section, we focus on the loss function \(\mathcal{L}(\cdot)\). Mimick-like models often adopt the mean squared error (MSE), which tries to give words with the same surface forms similar embeddings. However, the MSE only pulls positive word pairs closer, and does not push negative word pairs apart. Therefore, we use the contrastive loss instead (Equation 3). Wang and Isola (2020) found that the contrastive loss optimizes two key properties: Alignment and Uniformity. The Alignment describes the expected distance (closeness) between positive pairs:

\[
\ell_{\text{align}} = \mathbb{E}_{(x,y)\sim p_{\text{pos}}} \psi(u_x, u_y)
\]  (6)

Here, \(p_{\text{pos}}\) is the distribution of positive pairs. The Uniformity measures whether the learned representations are uniformly distributed in the hypersphere:

\[
\ell_{\text{uniform}} = \log \mathbb{E}_{(x,y)\sim p_{\text{data}}} e^{-t \psi(u_x, u_y)}
\]  (7)

Here, \(p_{\text{data}}\) is the data distribution and \(t > 0\) is a parameter. The two properties are consistent with our expected word representations: positive word pairs should be kept close and negative word pairs should be far from each other, finally scattered over the hypersphere.

4.4 Data Augmentation and Hard Negatives

Our positive word pairs are generated by data augmentation, which can increase the amount of training samples by using existing data. We use various strategies (Figure 4) to increase the diversity of our training samples: (1) Swap two adjacent characters, (2) Drop a character, (3) Insert a new character, (4) Replace a character according to keyboard distance, (5) Replace the original word by a synonymous word. The first four augmentations are originally designed to protect against adversarial attacks (Pruthi et al., 2019). We add the synonym replacement strategy to keep semantically similar words close in the embedding space – something that cannot be achieved by the surface form alone. Specifically, a set of synonyms is obtained by retrieving the nearest neighbors from pre-trained embeddings like FastText.
Negative word pairs are usually chosen randomly from the mini-batch. However, we train our model to be specifically resilient to hard negatives (or difficult negatives), i.e., words with similar surface forms but different meanings (e.g., misspelling and dispelling). To this end, we add a certain number of hard negative samples (currently 3 of them) to the mini-batch, by selecting word pairs that are not synonyms and have a small edit distance.

4.5 Mimicking Dynamical Embeddings

Pre-trained Language Models (e.g., ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019)) dynamically generate word representations based on specific contexts, which cannot be mimicked directly. To this end, we have two options: We can either learn the behavior of the input embeddings in BERT before the multi-layer attentions or mimic the static distilled embeddings (Bommasani et al., 2020; Gupta and Jaggi, 2021).

We use the BERT as an example to explain these two methods. Suppose we have a subword sequence after applying WordPiece to a sentence: \( W = \{w_1, w_2, ..., w_n\} \). For the subword sequence \( W \), BERT first represents it as a list of subword embeddings: \( E^{in} = \{e^{in}_{1}, e^{in}_{2}, ..., e^{in}_{n}\} \). We refer to this static representation as the Input Embedding of BERT, and we can use our model to mimic the behavior of this part. We call this method mimicking input embeddings. For ease of implementation, we learn only from the words that are not separated into pieces. After that step, BERT applies a multi-layer multi-head attention to the input embeddings \( E^{in} \), which yields a contextual representation for each subword: \( E^{out} = \{e^{out}_{1}, e^{out}_{2}, ..., e^{out}_{n}\} \). However, these contextual representations vary according to the input sentence and we cannot learn from them directly. Instead, we choose to mimic the distilled static embeddings from BERT, which are obtained by pooling (max or average) the contextual embeddings of the word in different sentences. We call this method mimicking distilled embeddings. The latter allows for better word representations, while the former does not require training on a large-scale corpus. Our empirical studies show that mimicking distilled embeddings performs only marginally better. Therefore, we decided to rather learn the input embeddings of BERT, which is simple yet effective.

4.6 Plug and Play

One of the key advantages of our model is that it can be used as a plug-in for other models. For models with static word embeddings like FastText, one can simply use our model to generate vectors for unseen words. For models with dynamic word embeddings like BERT, if a single word is tokenized into several parts, e.g., misspelling = \{miss, #pel, #ling\}, we regard it as an OOV word. Then, we replace the embeddings of the subwords by a single embedding produced by our model before the attention layer. Our final enhanced BERT model has 768 dimensions and 16M parameters. Note that the BERT-base model has ~110M parameters and its distilled one has ~550M parameters.

5 Experiments

5.1 Evaluation Datasets

There are two main methods to evaluate word representations: Intrinsic and Extrinsic. Intrinsic evaluations measure syntactic or semantic relationships between words directly, e.g., word similarity in word clusters. Extrinsic evaluations measure the performance of word embeddings as input features to a downstream task, e.g., named entity recognition (NER) and text classification. Several studies have shown that there is no consistent correlation between intrinsic and extrinsic evaluation results (Chiu et al., 2016; Faruqui et al., 2016; Wang et al., 2019). Hence, we evaluate our representation by both intrinsic and extrinsic metrics. Specifically, we use 8 intrinsic datasets (6 word similarity and 2 word cluster tasks): RareWord (Luong et al., 2013), SimLex (Hill et al., 2015), MTurk (Halawi et al., 2012), MEN (Bruni et al., 2014), WordSim (Agirre
et al., 2009), Simverb (Agirre et al., 2009), AP (Almuhareb, 2006) and BLESS (Baroni and Lenci, 2011). We use four extrinsic datasets (2 text classification and 2 NER tasks): SST2 (Socher et al., 2013), MR (Pang and Lee, 2005), CoNLL-03 (Sang and De Meulder, 2003) and BC2GM (Smith et al., 2008). It is worth noting that the RareWord dataset contains many long-tail words and the BC2GM is a domain-specific NER dataset. All data augmentations and typo simulations are implemented by NLPAUG\(^1\). Appendix B provides more details on our datasets and experimental settings.

5.2 Results on Intrinsic Tasks

Table 2 shows the experimental results on 8 intrinsic tasks. Compared to other mimick-like models, our model achieves the best average score across 8 datasets while using the least number of parameters. Specifically, our model performs best on 5 word similarity tasks, and second-best on the word cluster tasks. Although there is a gap between our model and the original FastText, we find our performance acceptable, given that our model is 100x times smaller.

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameters</th>
<th>SST2</th>
<th>MR</th>
<th>CoNLL-03</th>
<th>BC2GM</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>embedding</td>
<td>others</td>
<td>+typo</td>
<td>original</td>
<td>+typo</td>
<td>original</td>
</tr>
<tr>
<td>FastText (2017)</td>
<td>969M</td>
<td>-</td>
<td>82.3</td>
<td>60.5</td>
<td>73.3</td>
<td>62.2</td>
</tr>
<tr>
<td>Edit Distance</td>
<td>969M</td>
<td>-</td>
<td>82.3</td>
<td>60.5</td>
<td>73.3</td>
<td>62.2</td>
</tr>
<tr>
<td>MIMICK (2018)</td>
<td>9M</td>
<td>517K</td>
<td>69.7</td>
<td>62.3</td>
<td>73.6</td>
<td>61.4</td>
</tr>
<tr>
<td>BoS (2018)</td>
<td>500M</td>
<td>-</td>
<td>72.7</td>
<td>72.6</td>
<td>73.8</td>
<td>69.5</td>
</tr>
<tr>
<td>KVQ-FH (2019)</td>
<td>12M</td>
<td>-</td>
<td>77.8</td>
<td>71.4</td>
<td>72.9</td>
<td>66.5</td>
</tr>
<tr>
<td>LOVE</td>
<td>6.3M</td>
<td>200K</td>
<td>81.4</td>
<td>73.2</td>
<td>74.4</td>
<td>66.7</td>
</tr>
</tbody>
</table>

Table 3: Performance on the extrinsic tasks, measured as accuracy and F1 (five runs of different learning rates) for text classification and NER, respectively. Typos are generated by simulated errors of an OCR engine (Ma, 2019). The speed of producing word vectors with Edit Distance and LOVE is 380s/10K words and 0.9s/10K words, respectively.

5.3 Results on Extrinsic Tasks

Table 3 shows the results on four downstream datasets and their corrupted version. In this experiment, we introduce another non-trivial baseline: Edit Distance. For each corrupted word, we find the most similar word from a vocabulary using edit distance and then use the pre-trained vectors of the retrieved word. Considering the time cost, we only use the first 20K words appearing in FastText (2M words) as reference vocabulary.

The typo words are generated by simulating post-OCR errors. For the original datasets, our model obtains the best results across 2 datasets and the second-best on NER datasets compared to other mimick-like models. For the corrupted datasets, the performance of the FastText model decreases a lot and our model is the second best but has very close scores with BoS consistently. Compared to other mimick-like models, our model with 6.5M achieves the best average score. Although Edit Distance can effectively restore the original meaning of word, it is 400x times more time-consuming than our model.

5.4 Robustness Evaluation

In this experiment, we evaluate the robustness of our model by gradually adding simulated post-OCR typos (Ma, 2019). Table 4 shows the performances on SST2 and CoNLL-03 datasets. We observe that our model can improve the robustness of the original embeddings without degrading their performance. Moreover, we find our model can make FastText more robust compared to other commonly used methods against unseen words: a generic UNK token or character-level representation of neural networks. Figure 5 shows the robust-
Table 4: Robust evaluation (five runs of different learning rates) on text classification and NER under simulated post-OCR typos. We use uncased and cased BERT-base model for SST2 and CoNLL-03, respectively.

<table>
<thead>
<tr>
<th>Typo Probability</th>
<th>original</th>
<th>10%</th>
<th>30%</th>
<th>50%</th>
<th>70%</th>
<th>90%</th>
<th>original</th>
<th>10%</th>
<th>30%</th>
<th>50%</th>
<th>70%</th>
<th>90%</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastText</td>
<td>82.3</td>
<td>68.2</td>
<td>59.8</td>
<td>56.7</td>
<td>57.8</td>
<td>60.3</td>
<td>86.4</td>
<td>81.6</td>
<td>78.9</td>
<td>73.9</td>
<td>70.2</td>
<td>63.4</td>
<td>70.0</td>
</tr>
<tr>
<td>FastText + LOVE</td>
<td>82.1</td>
<td>79.8</td>
<td>74.9</td>
<td>74.2</td>
<td>68.8</td>
<td>67.2</td>
<td>86.3</td>
<td>84.7</td>
<td>81.8</td>
<td>77.5</td>
<td>73.1</td>
<td>71.3</td>
<td>76.8</td>
</tr>
<tr>
<td>BERT</td>
<td>91.5</td>
<td>88.2</td>
<td>79.9</td>
<td>74.7</td>
<td>69.0</td>
<td>60.1</td>
<td>91.2</td>
<td>89.8</td>
<td>86.2</td>
<td>83.4</td>
<td>79.9</td>
<td>76.5</td>
<td>80.7</td>
</tr>
<tr>
<td>BERT + LOVE</td>
<td>91.5</td>
<td>88.3</td>
<td>83.7</td>
<td>77.4</td>
<td>72.7</td>
<td>63.3</td>
<td>89.9</td>
<td>88.3</td>
<td>86.1</td>
<td>84.3</td>
<td>80.8</td>
<td>78.3</td>
<td>82.1</td>
</tr>
</tbody>
</table>

Table 5: Ablation studies for the architecture of LOVE, measured as Spearman’s ρ and accuracy, respectively.

<table>
<thead>
<tr>
<th>Ablation of data augmentation and hard negatives</th>
<th>parameters</th>
<th>RareWord</th>
<th>SST2</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/out hard negatives w/out hard negatives and augmentation</td>
<td>6.3M</td>
<td>200K</td>
<td>37.7</td>
</tr>
<tr>
<td>w/out hard negatives</td>
<td>6.3M</td>
<td>200K</td>
<td>34.5</td>
</tr>
<tr>
<td>w/out hard negatives and augmentation</td>
<td>6.3M</td>
<td>200K</td>
<td>40.8</td>
</tr>
</tbody>
</table>

5.5 Ablation Study

We now vary the components in our architecture (input method, encoder and loss function) to demonstrate the effectiveness of our architecture.

Input Method. To validate the effect of our Mixed Input strategy, we compare it with two other methods: using only the character sequence or only the subword sequence. Table 5 shows that the Mixed method achieves better representations, and any removal of char or subword information can decrease the performance.

Encoder. To encode the input sequence, we developed the Positional Attention Module (PAM), which first extracts ngram-like local features and then uses self-attention combine them without distance restrictions. Table 5 shows that PAM performs the best, which validates our strategy of incorporating both local and global parts inside a word. At the same time, the number of parameters of PAM is acceptable in comparison. We visualize the attention weights of PAM in Appendix C.4, to show how the encoder extracts local and global morphological features of a word.

Loss Function. LOVE uses the contrastive loss, which increases alignment and uniformity. Wang and Isola (2020) proves that optimizing directly these two metrics leads to comparable or better performance than the original contrastive loss. Such a loss function can be written as:

$$\ell_{au} = \ell_{align} + \lambda \cdot \ell_{uniform}$$ (8)

Here, $\lambda$ is a hyperparameter that controls the impact of $\ell_{uniform}$. We set this value to 1.0 because it achieves the best average score on RareWord and SST2. An alternative is to use the Mean Squared Error (MSE), as in mimick-like models. Table 5 compares the performances of these different loss functions. The contrastive loss significantly outperforms the MSE, and there is no obvious improve-
Table 6: Performances of different strategies that work with BERT together, measured as the accuracy among five different learning rates.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Typo-0</th>
<th>Typo-1</th>
<th>Typo-2</th>
<th>Typo-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SST2</td>
<td>91.5</td>
<td>77.2</td>
<td>73.2</td>
<td>69.4</td>
</tr>
<tr>
<td>Mimicking Input Embeddings</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT</td>
<td>91.3</td>
<td>77.2</td>
<td>73.5</td>
<td>70.7</td>
</tr>
<tr>
<td>BERT + Add</td>
<td>91.4</td>
<td>79.6</td>
<td>77.2</td>
<td>72.8</td>
</tr>
<tr>
<td>BERT + Linear (2020)</td>
<td>91.5</td>
<td>81.4</td>
<td>78.7</td>
<td>73.6</td>
</tr>
<tr>
<td>Mimicking Distilled Embeddings</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT + Add</td>
<td>91.3</td>
<td>78.8</td>
<td>75.6</td>
<td>72.3</td>
</tr>
<tr>
<td>BERT + Linear (2020)</td>
<td>91.3</td>
<td>81.4</td>
<td>78.7</td>
<td>73.6</td>
</tr>
<tr>
<td>BERT + Replacement</td>
<td>91.4</td>
<td>81.5</td>
<td>78.9</td>
<td>73.8</td>
</tr>
</tbody>
</table>

5.6 The performance of mimicking BERT

As described in Section 4.5, we can mimic the input or distilled embeddings of BERT. After learning from BERT, we use the vectors generated by LOVE to replace the embeddings of OOV subwords. Finally, these new representations are fed into the multi-layer attentions. We call this method the Replacement strategy. To validate its effectiveness, we compare it with two other baselines: (1) Linear Combination (Fukuda et al., 2020). For each subword $e_{sub}$, the generated vectors of word $e_{word}$ containing the subwords are added to the subword $e_{new} = (1 - \alpha) e_{sub} + \alpha e_{word}$

\[ \alpha = \text{sigmoid} (W \cdot e_{sub}) \]

where $e_{sub} \in \mathbb{R}^d$ is a subword vector of BERT, and $e_{word} \in \mathbb{R}^d$ is a generated vector of our model. $W \in \mathbb{R}^d$ are trainable parameters. (2) Add. A generated word vector is directly added to a corresponding subword vector of BERT:

\[ e_{new} = e_{sub} + e_{word} \]

Table 6 shows the result of these strategies. All of them can bring a certain degree of robustness to BERT without decreasing the original capability, which demonstrates the effectiveness of our framework. Second, the replacement strategy consistently performs best. We conjecture that BERT cannot restore a reasonable meaning for those rare and misspelling words that are tokenized into subwords, and our generated vectors can be located nearby the original word in the space. Third, we find mimicking distilled embeddings performs the best while mimicking input embeddings comes close. Considering that the first method needs training on large-scale data, mimicking the input embeddings is our method of choice.

6 Conclusion

We have presented a lightweight contrastive-learning framework, LOVE, to learn word representations that are robust even in the face of out-of-vocabulary words. Through a series of empirical studies, we have shown that our model (with only 6.5M parameters) can achieve similar or even better word embeddings on both intrinsic and extrinsic evaluations compared to other mimick-like models. Moreover, our model can be added to models with static embeddings (such as FastText) or dynamical embeddings (such as BERT) in a plug-and-play fashion, and bring significant improvements there. For future work, we aim to extend our model to languages other than English.

7 Acknowledgements

We sincerely thank all the reviewers for their insightful comments and helpful suggestions. This work was partially funded by ANR-20-CHIA-0012-01 (“NoRDF”).


Bin Wang, Angela Wang, Fenxiao Chen, Yuncheng Wang, and C-C Jay Kuo. 2019. Evaluating word embedding models: Methods and experimental results. APSIPA transactions on signal and information processing, 8.


We consider the following four methods to reduce the total parameters of our model:

1. **Matrix Decomposition.** The original matrix can be decomposed into two smaller matrices $V = U \times M$, $U \in \mathbb{R}^{V \times h}$, $M \in \mathbb{R}^{h \times m}$ and $h < m$. Here, we set $m = 300$ and $h = 200$ respectively.

2. **Top Subword.** We use only the top-$k$ frequent subwords, using the word frequencies from a corpus. We set the parameter $k = 20000$.

3. **Hashing.** We use a hashing strategy to share memory for subwords aiming to reduce the parameters. We use a bucket size of 20000.

4. **Preprocessing.** The original vocabulary contains plurals and conjugations, therefore we stem all complete words and remove words with numerals, obtaining a new vocabulary of 21257 words.

Table A1 shows that the preprocessing method can reduce parameters very effectively while obtaining a very competitive performance.

<table>
<thead>
<tr>
<th>Hyperparam</th>
<th>SST2</th>
<th>MR</th>
<th>CONLL-03</th>
<th>BC2GM</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>CNN</td>
<td>CNN</td>
<td>BiLSTM+CRF</td>
<td>BiLSTM+CRF</td>
</tr>
<tr>
<td>layer</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>kernel</td>
<td>(3,4)</td>
<td>(3,4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>filter</td>
<td>100</td>
<td>100</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>hidden size</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>optimizer</td>
<td>Adam</td>
<td>Adam</td>
<td>SGD</td>
<td>SGD</td>
</tr>
<tr>
<td>dropout</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>batch size</td>
<td>50</td>
<td>50</td>
<td>768</td>
<td>768</td>
</tr>
<tr>
<td>epoch</td>
<td>5</td>
<td>5</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table A2: Hyperparameters for extrinsic datasets.

As for our model, we first lower-case and tokenize each word by using WordPiece (Wu et al., 2016) with a vocabulary of $30K$ subwords and preprocess them by stemming and removing subwords with numerals. This yields a vocabulary of 21257 words. Each subword is represented by corresponding vectors from our model and we adopt a modified attention model to encode the subword sequence. Specifically, the layer number of this encoder is just 1 for efficiency and the hidden dimension is 300. In each block, the number of attention heads is 1 and we use fixed sinusoidal position embeddings (Vaswani et al., 2017) for positional information. To train the contrastive learning framework, we use the open-source tool (Ma, 2019) to augment a word, and use the probabilities $\{0.07, 0.07, 0.07, 0.07, 0.36, 0.36\}$ for six augmentations: swap, drop, insert, keyboard, synonym, no-operation. Hard negatives are generated by edit distance. For each target word, we store the top-100 similar words and insert 3 of them into a mini-batch as hard negatives. The loss function is a standard contrastive loss with temperature $\tau = 0.07$. The optimizer is Adam and the learning rate is 0.002. The dropout rate is 0.2 and we train the model for 20 epochs in total.
Figure A2: PCA visualizations of word vectors generated by LOVE, BoS, and KVQ-FH. Different colors mean different clusters, as predicted by K-means. There are three OOV words: oxygen, architect and leukemia.

Figure A3: Performances of different augmentations on RareWord, measured as Spearman’s ρ. Diagonal entries correspond to individual augmentation and off-diagonal entries correspond to composite augmentation.

Figure A4: Performances of different augmentations on SST2, measured as accuracy. Diagonal entries correspond to individual augmentation and off-diagonal entries correspond to composite augmentation.

**B.3 Intrinsic and Extrinsic Evaluations**

We choose the setting discussed in Section 4 to train our model for 20 epochs, and evaluate each intrinsic task based on the vectors that the models produce. As for the extrinsic tasks, we feed word vectors into each neural network and fix them during training. We use CNNs for text classification (Zhang and Wallace, 2015) and BiLSTM+CRF for NER (Huang et al., 2015). We compare different embeddings on both intrinsic and extrinsic datasets by using generated vectors. For the word cluster tasks, the produced vectors are clustered by K-Means and then measured by Purity. The hyper-parameters of the extrinsic tasks are shown in Table A2. For each dataset, our model is trained with five learning rates $\{5e^{-3}, 3e^{-3}, 1e^{-3}, 8e^{-4}, 5e^{-4}\}$. We select the best one on the development set to report its score on the test set.

To generate a corrupted dataset, we simulate post-OCR errors. We adopt the augmentation tool developed by Ma (2019) to corrupt 70% of the original words. To check the robustness of BERT, we directly finetune a BERT-base model using Huggingface (Wolf et al., 2020). During finetuning, the batch size is 16 and we train 5 epochs. We select the best model among five learning rates $\{9e^{-5}, 7e^{-5}, 5e^{-5}, 3e^{-5}, 1e^{-5}\}$ on the development set and report the score of the model on the test set.

**B.4 Datasets**

**Intrinsic Datasets.** We use six word similarity datasets: (1) RareWord (Luong et al., 2013) (2) SimLex (Hill et al., 2015) (3) MTurk (Halawi et al., 2012) (4) MEN (Bruni et al., 2014) (5) Word-
Sim (Agirre et al., 2009), and (6) Simverb (Agirre et al., 2009). The task is scored by Spearman’s \( \rho \), which computes the correlation between gold similarity and the similarity obtained from generated vectors. For the word cluster task, we use (1) AP (Almuhareb, 2006) and (2) BLESS (Baroni and Lenci, 2011). The generated word vectors are first clustered by K-means (MacQueen et al., 1967) and then scored by the cluster purity.

**Extrinsic Datasets.** We use both sentence-level and token-level downstream datasets to evaluate the quality of word representations. For the sentence level, we use SST2 (Socher et al., 2013) and MR (Pang and Lee, 2005), and the metric is accuracy. For the token level, we use two NER datasets: general CoNLL-03 (Sang and De Meulder, 2003) and biomedical BC2GM (Smith et al., 2008). The metric is the entity-level F1 score. As before, we select the best model among five different learning rates \{5e-3, 3e-3, 1e-3, 8e-4, 5e-4\} on the development set and then report the model score on the test set.

### C Additional Analyses

#### C.1 Qualitative Analysis

To better understand the clusterings produced by LOVE, we chose 15 words from the AP dataset (Almuhareb, 2006), covering three topics (Chemical Substance, Illness, and Occupation). We added 3 corrupted words, oxygen, architect and leukemia. Figure A2 shows how LOVE, BoS, and KVQ-FH cluster these words (using a PCA projection and K-means). All approaches space out the clusters to some degree. In particular, BoS and KVQ-FH have trouble separating professions and chemical substances. For the corrupted words, only LOVE is able to embed them close enough to their original form, so that they appear in the correct cluster.

#### C.2 Effect of Augmentation for Text Classification

Figure A4 shows the performance of five augmentation strategies on the text classification task SST2. We observe that synonym is the most effective methods. The first four methods have a weaker effect, but the keyboard replacement can bring a certain degree of improvement. The results on RareWord are similar (Figure A3).

#### C.3 Effect of \( \tau \) in Contrastive Loss

As discussed in Chen et al. (2020), a proper temperature can yield better representation in the visual area because \( \tau \) is able to weigh the negatives by their relative hardness. As shown in Table A3, we attempt different values of temperature and find that there is no consistent \( \tau \) that makes a model work well both on intrinsic and extrinsic datasets. Hence, we choose the best performer on average, i.e., \( \tau = 0.07 \).

#### C.4 Visualization of Encoder

As mentioned before, we combine two types of attention heads (self-attention and positional attention) to encode a subword sequence. Here, we vi-
<table>
<thead>
<tr>
<th>parameters</th>
<th>embedding</th>
<th>non-embedding</th>
<th>RareWord</th>
<th>SST2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_{cl}$ ($\tau = 0.03$)</td>
<td>9M</td>
<td>200K</td>
<td>35.0</td>
<td><strong>81.6</strong></td>
</tr>
<tr>
<td>$\ell_{cl}$ ($\tau = 0.07$)</td>
<td>9M</td>
<td>200K</td>
<td>39.0</td>
<td>81.3</td>
</tr>
<tr>
<td>$\ell_{cl}$ ($\tau = 0.12$)</td>
<td>9M</td>
<td>200K</td>
<td>39.9</td>
<td>81.1</td>
</tr>
<tr>
<td>$\ell_{cl}$ ($\tau = 0.20$)</td>
<td>9M</td>
<td>200K</td>
<td>37.6</td>
<td>81.5</td>
</tr>
<tr>
<td>$\ell_{cl}$ ($\tau = 0.50$)</td>
<td>9M</td>
<td>200K</td>
<td>38.1</td>
<td>80.6</td>
</tr>
</tbody>
</table>

Table A3: Performances of contrastive loss with various temperature $\tau$, measured as Spearman’s $\rho$ and accuracy respectively.

visualize the attention weights on each side and show how they work. Figure A5 shows the position-dependent weights. We use sinusoidal functions to generate positional embeddings, and the weights are the dot product between these embeddings. We observe the positional weights tend to the left and right subwords in addition to themselves, which yields trigram representations.

Figure A6 shows the self-attention weights which are computed from the trigram subwords of positional attention. Hence, each subword in this figure is a trigram representation instead of a single subword representation. As we see, self-attention can capture global features regardless of distance. We take the first token $[CLS]$ as an example, and this self-attention assigns high weights for the token $e$ and $[SEP]$, which constructs a representation like this: $[CLS] b + me[SUB] + #me[SEP]$. This segment tells us this word starts with $b$ and ends with $me$. 