Analogy-Guided Evolutionary Pretraining of Binary Word Embeddings

R. Alexander Knipper[*], **Md. Mahadi Hassan**[*], **Mehdi Sadi**[†], **Shubhra Kanti Karmaker Santu**[*]
BDI Lab, Department of Computer Science & Software Engineering[*]
Department of Electrical & Computer Engineering[†]
Auburn University, Alabama, USA
{rak0035, mzh0167, mzs0190, sks0086}@auburn.edu

Abstract

As we begin to see low-powered computing paradigms (Neuromorphic Computing, Spiking Neural Networks, etc.) becoming more popular, learning binary word embeddings has become increasingly important for supporting NLP applications at the edge. Existing binary word embeddings are mostly derived from pretrained real-valued embeddings through different simple transformations, which often break the semantic consistency and the so-called "arithmetic" properties learned by the original, real-valued embeddings. This paper aims to address this limitation by introducing a new approach to learn binary embeddings from *scratch*, preserving the semantic relationships between words as well as the arithmetic properties of the embeddings themselves. To achieve this, we propose a novel genetic algorithm to learn the relationships between words from existing word analogy data-sets, carefully making sure that the arithmetic properties of the relationships are preserved. Evaluating our generated 16, 32, and 64-bit binary word embeddings on Mikolov's word analogy task shows that more than 95% of the time, the best fit for the analogy is ranked in the top 5 most similar words in terms of cosine similarity.

1 Introduction

Word embeddings see very common use in many widely-adopted NLP applications, e.g., document summarization (El-Kassas et al., 2021), sentiment analysis (Yadav and Vishwakarma, 2020), entity extraction (Li et al., 2020), question answering (Jin et al., 2022), etc. However, the majority of commonly-used word embeddings are far too demanding in terms of energy and computational resources required to train and load them, making state-of-the-art word embeddings unsuitable for use in a low-energy environment, like in an internet of things (IoT) device (Zadeh et al., 2020; Wang et al., 2020; Daghero et al., 2021) or in a Neuromorphic processor (Schuman et al., 2022; Davies et al., 2021). As we observe these low-powered devices entering the mainstream, we become increasingly aware of our inability to use typical word embeddings in those environments, since typical word embeddings usually require multiple gigabytes for storage and hundreds (if not thousands) of floating-point multiplications to capture meaningful relationships between words. Furthermore, low-energy neuromorphic computers in particular are based on binary "spiking" inputs and perform calculations using "accumulation" (sum) operations, therefore *not* supporting floating-point operations (Poon and Zhou, 2011; Davies et al., 2021). Hence, real-valued embeddings are of little use to these low-energy computing paradigms, which is our main motivation for learning high-quality binary embeddings.

An intuitive way to address this issue is to take the pretrained real-valued word embeddings and directly *binarize* them so they can be easily used as a spike train for input to a neuromorphic processor. The potential benefits of this approach are astronomical, as the vector's size can be reduced by more than 95% and the number of operations needed goes down significantly (Tissier et al., 2019). As an example, calculating the similarity between two words goes from requiring $O(n)$ floating-point operations to 2 binary operations: an *XNOR* and a *bit-count* operation. However, one of the primary issues to address when binarizing word embeddings is making sure that this oversimplification in word representation does not cause a significant drop in semantic and syntactic accuracy of the learned embeddings. In other words, binary embeddings still need to encode semantic and syntactic information properly so that meaningful relations can be captured when these embeddings are used.

The simplest approach for creating binary embeddings is to quantize the real-valued embedding vectors into binary labels based on some thresh-

olds (Faruqui et al., 2015). While the thresholding approach is simple, it often breaks the semantic relationships learned by the real-valued vectors, as an infinite range of real-valued numbers are forced to map into one/zero labels without considering the loss in semantic consistency during the process. Another approach is to learn an auto-encoder which can transform a real-valued embedding vector into a binary vector while minimizing the loss in semantic consistency during the process (Tissier et al., 2019). However, this process still assumes that high-quality, real-valued embeddings are already available, and their experimental results show that the binary embeddings learned this way fail to achieve comparable performance against real-valued embeddings in both Semantic and Syntactic Analogy tasks (Tissier et al., 2019).

To address these limitations, we propose to learn binary embeddings from *scratch*, which will guarantee the preservation of the semantic and syntactic relationships between words even in the restricted binary latent space. Our primary motivation for proposing this method is to take a step forward towards enabling NLP in the emerging low-power neuromorphic computing paradigm. We envision using this binary embedding as an encoded input to Spiking Neural Networks (SNNs), providing a compact spike-representation for words to be processed in downstream NLP tasks. Furthermore, since SNNs currently have difficulty learning with methods supported by backpropagation (Luo et al., 2022), we opt to utilize a method that has no need for it, i.e., a *Genetic Algorithm* (Holland, 1992).

Genetic algorithms are a class of methods for solving both constrained and unconstrained optimization problems based on the concept of "survival of the fittest" (Holland, 1992), and naturally they fit binary representations intuitively because of the crossover and mutation operators associated with them (Katoch et al., 2021). However, one common criticism of genetic algorithms is their slow convergence (Vie et al., 2020). We propose to address this limitation by designing an objective function which is guided by high quality analogy examples to facilitate faster convergence. To be more specific, in a typical embedding training process, the final encoding is learned by observing word co-occurrences (Mikolov et al., 2013b), which is often very noisy. In contrast, our proposed method learns this encoding using a data-set of high-quality targeted analogies, allowing for a more focused understanding of the relationships between words in a curated vocabulary and a faster convergence while training. This becomes especially useful for IoT applications, where a full vocabulary may not be necessary as opposed to a smaller collection of relevant words. Another major benefit of the proposed approach is that it can learn the goal "binary" embeddings without worrying about the hassles of implementing backpropagation in spiking neural networks.

Experiments with the evolved 16-, 32-, and 64-bit binary word embeddings on the word analogy task (Mikolov et al., 2013a) show that more than 95% of the time, the best fit for the analogy is ranked among the top 5 most similar words in the vocabulary. This demonstrates that the proposed technique is effective as well as useful.

The rest of the paper is organized as follows: Section 2 presents the related works. Next, Section 3 provides some basic background on genetic algorithms and evolutionary operators. Section 4 presents the details of the proposed evolutionary training process followed by our experimental setup (Section 5) and experimental results (Section 6). Finally, we conclude the paper in Section 7.

2 Related Works

2.1 Language Modeling

Word Embeddings: Classical word embeddings capture semantic and syntactic information by observing word co-occurrences and predicting either the target word or the context given the other one (Mikolov et al., 2013a). This helps learn relationships among words that, surprisingly enough, can be largely represented with arithmetic expressions of the word vectors themselves. These models are then improved upon with the introduction of Negative Sampling as a replacement to the hierarchical Softmax layers originally used by (Mikolov et al., 2013b). Another option for learning word embeddings is to utilize global word co-occurrence counts (Pennington et al., 2014), on the intuition that the ratios between word co-occurrences encode more information than the raw co-occurrence counts, which results in the commonly-used word embedding, GloVe.

Contextual Word Embeddings: Contextual word embeddings can encode how the meaning of a word changes with its context (Peters et al., 2018). As context-based embeddings gained traction, we be-

gan to see their use as a part of transformer architectures (Devlin et al., 2019; Lewis et al., 2019). However, encoding contextual information to that extent sits outside the current scope of this paper.

N-Gram and Sentence Embeddings: Beyond word embeddings, researchers have proposed methods that encode larger language constructs, such as n-grams (Bojanowski et al., 2017) or sentences (Conneau et al., 2017; Cer et al., 2018), but those are also beyond the scope of this paper as we focus exclusively on word embeddings.

2.2 Efficient NLP

Over the past few years, some NLP research has trended towards making existing methods more efficient, but these research directions primarily focus on distilling transformer architectures to get a similar-performing, smaller model (Sanh et al., 2019; Jiao et al., 2019; Sun et al., 2020; Iandola et al., 2020). While these advances help improve the efficiency of contextualized word embeddings, they do not help in the case of binary representations.

Neuromorphic Computing Neuromorphic computing is a relatively newer field, providing incredibly low-powered hardware with a new architecture called a spiking neural network (SNN) (Poon and Zhou, 2011; Davies et al., 2021; Roy et al., 2019). At present, SNNs are difficult to train, *exclusively* requiring spike inputs and lacking support for typical backpropagation and commonly-used activation functions. As a result, the common workaround thus far is to train a neural network in the real-valued domain and convert it to a spiking neural network (Sengupta et al., 2019). The advances made in SNNs so far have mainly been in computer vision (Kim and Panda, 2021) and signal processing (Auge et al., 2021), but it shows promise as a wide-use field for efficient, powerful learning.

Binary Word Embeddings: To execute NLP tasks in the *Neuromorphic Computing* paradigm, we need to provide binary/spike inputs. This is where binarization techniques become relevant. (Joulin et al., 2016) proposed a hash-based clustering technique for learning binary embeddings, where they concatenated the binary codes of the closest centroids for each word. Another method is to transform an existing real-valued embedding to a binary embedding using an auto-encoder (Tissier et al., 2019), and yet another method is to learn correlations between one-hot encoded context and target blocks (Liang et al., 2021).

2.3 Genetic Algorithms

Genetic algorithms (Holland, 1992) often find use in solving optimization problems for which an exact mathematical problem definition is either difficult to create or cannot be calculated given the problem constraints (Sivanandam and Deepa, 2008). However, due to their general ease of use in solving optimization problems, they find some use in recent NLP research (Karcioğlu and Yaşa, 2020; Ince, 2022). More details are provided in Section 3.

2.4 Difference From Previous Work

Our approach, in contrast to previous word embedding binarization methods, aims to learn word embeddings from *scratch* for use in downstream applications in SNNs. In order to best adhere to that end, we opt to *not employ backpropagation*, making our problem a bit more difficult to solve with classical methods. Due to that, we decide to utilize a genetic algorithm to generate binary embeddings, framed as a problem of optimizing how much semantic/syntactic information it can encode from a curated set of analogistic relationships.

3 Background on Genetic Algorithm

Genetic Algorithms are a family of computational models inspired by evolution (Kumar et al., 2018). These algorithms encode a potential solution to a specific problem through a simple chromosome-like data structure and apply recombination operators to these structures so as to preserve critical information. Genetic algorithms are often viewed as function approximators, although the range of problems to which evolutionary algorithms have been applied is quite broad (Deb, 2011).

An implementation of a *Genetic Algorithm* begins with a population of (typically random) chromosomes. One then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes which represent a better solution to the target problem are given more chances to reproduce than the chromosomes which are poorer solutions. The "goodness" of a solution is typically defined w.r.t. the current population.

3.1 The Terminologies

A few terms must be explained before we go into our proposed algorithm in detail.

Population: The *population* contains μ candidate solutions. The key idea here is to update this population iteratively so that one can end up with the best solution. The initial population contains near-random solutions, and the goal of the population is to evolve a better solution over time using genetic recombination operators.

Individual and Allele: Each of the μ members of the population is referred to as an *individual* or *chromosome*. Each individual consists of a number of attributes, called genes. Each gene in turn may be associated with some values, which are called *alleles*. Alleles are optional and not always present.

Fitness Function: There is a function which *evaluates* an individual, i.e. assigns a score on the basis of how "good" it is. Therefore, this function assigns higher scores for "good" individuals and lower scores for "bad" individuals. This function is known as a *fitness function*.

3.2 Evolutionary Operators

The operators in an evolutionary algorithm are quite similar to biological evolution in nature. A brief overview of the operators is as follows.

Selection: The idea of selection is to pick chromosomes from the population that have the best chance at improving the overall fitness of the population in the next iteration. To achieve this, $(1-r)\mu$ individuals from the *best* individuals in the population are chosen, where r is the fractional number of chromosomes to be replaced at each step. How do we sort out the best individuals? The idea is simple, based on a threshold called the *fitness threshold*. The fitness threshold works as a filter: chromosomes with fitness values higher than this threshold are considered to be in the next generation, while the lower values are discarded. However, fitness thresholds are not always present, such as in *Roulette Wheel Selection* (used in this work), to be described next.

Roulette Wheel Selection: In *Roulette Wheel Selection* (Lloyd and Amos, 2017), no individuals are discarded directly regardless of their fitness scores. Rather, the normalized fitness score of individual i is returned by the fitness function, as indicated by equation 2, and selection is done in a probabilistic fashion using the following formula.

$$p_i = \frac{f_i}{\sum_{j=1}^{|P|} f_j} \tag{1}$$

Where P is the population, p_i is the probability of chromosome i being selected, and both f_i and f_j are the fitness of chromosome i or j, respectively.

Tournament Selection: In *Tournament Selection* (Butz et al., 2003), two individuals are first chosen at random from the current population. With some predefined probability p, the higher-scoring individual of these two is selected, and with probability $(1-p)$, the lower-scoring individual is selected.

Crossover: For crossover, a pair of individuals are chosen according to a predefined selection strategy. For each selected pair, a new pair is generated by the crossover operator. The newly generated offspring pairs are added to the new population (Pavai and Geetha, 2016). Below, We discuss some variants of crossover.

Single Point Crossover: It is the simplest form of crossover, where the first n bits of the first offspring come from the first parent, followed by bits from the second parent. Similarly, the second offspring consists of bits from the second parent followed by bits from the first.

Two Point Crossover: Two point crossover works exactly like single point, except for one key difference. Here, the first few bits of the first offspring come from the second parent. Then, a few bits from the first parent are present, followed by more bits from the second parent, terminating the string.

Uniform Crossover: A more complicated version is uniform crossover, where each bit in each offspring can come from any parent with a particular probability, which is defined by the user.

Mutation: Mutation is an operator which alters one or more gene values with a small probability (Hall et al., 2020). It is used to maintain diversity in the solution, since as the algorithm converges we have no way of knowing whether we have found a local optima or the global optima. Mutation is used to help alleviate this problem, creating diversity in the solution space (Do et al., 2021).

4 Evolutionary Pretraining of Binary Word Embeddings

In this section, we describe the details of the evolutionary pretraining process to learn binary word embeddings from *scratch* which is guided by a collection of word analogy examples.

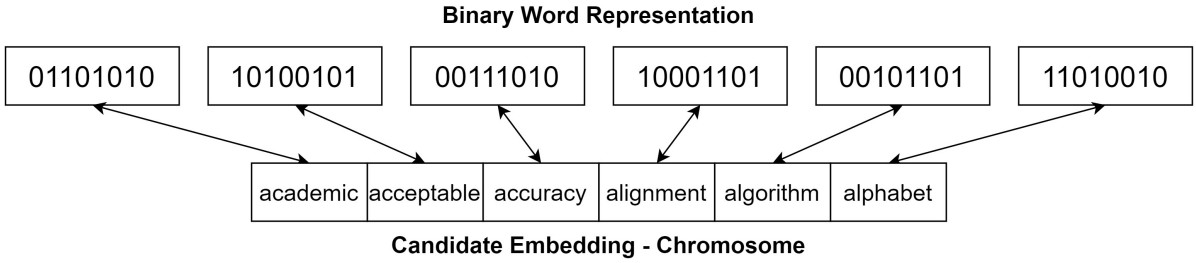

Figure 1: Chromosome Representation

4.1 Chromosome Representation and Initialization

To start, we initialize a population, P, containing μ individual chromosomes, where μ is a configurable parameter. As shown in Figure 1, each chromosome is a candidate solution, i.e., a *full* set of word embeddings for the given vocabulary. Each gene inside a chromosome represents a unique word from the vocabulary and each gene/word consists of d alleles (d is a user-defined hyper-parameter). Here, each allele is essentially a bit of the binary word embedding vector. Therefore, the chromosome is essentially a sequence of words where each word is a bit vector.

Chromosomes are constructed by randomly initializing a binary vector of dimension d for each word in the entire vocabulary, V. This results in a chromosome with a total length of $(V \times d)$ bits for evolutionary learning.

4.2 Evaluation and Fitness Function

Appropriate evaluation of a chromosome requires designing an accurate fitness function, which can measure the *goodness* of a candidate solution. Fitness functions are central components of evolutionary learning and are often the most challenging task. Indeed, *when can we say that an embedding is good/bad?* One option is to use the embedding for a downstream NLP task and measure the accuracy for that task as the fitness of the chromosome/candidate embedding. However, such indirect evaluation results may not hold in general for other downstream NLP tasks. Another option is to use the embedding for a wide variety of downstream tasks and compute their average accuracy as the fitness score. However, computing the fitness score in this fashion will be very time-consuming for an evolutionary algorithm to converge, as thousands of evaluations are needed to find a reasonably "good" solution and hence, it is impractical.

To address this challenge, we propose to evaluate chromosomes in terms of their capability to capture the the semantic/syntactic relationships between words explicitly using a set of word analogy examples. Mathematically, we evaluate each chromosome with the following fitness function, F.

$$F = \sum_{a_i \in A} BitCount(\{(a_i[1] \oplus a_i[2]) \vee a_i[3]\} \odot a_i[4])) \tag{2}$$

Here, A is the set of word analogy examples, with each analogy a_i having four words in the form $first - second + third = fourth$, and the $BitCount()$ operation counts the number of bits that are set to 1. The intuition behind this fitness function is primarily to enforce additive compositionality, as described in (Mikolov et al., 2013b), between the learned binary vectors, with the *XOR* operation ($\oplus$) serving as our bit-wise "subtraction" operation and the *OR* operation ($\vee$) serving as our bit-wise "addition" operation. The intuition behind these choices are as follows: the *XOR* operation outputs 1 when the input bits are different, therefore, it can serve as a proxy for the difference between two inputs in the binary domain. Similarly, we use the bit-wise *OR* operation to serve as a proxy for addition in the binary space. By comparing how closely the composition of the first three words ($first - second + third$) approximates the representation of the fourth word in the analogy (calculated by *XNOR*-ing the composition of the first three words and the fourth word), we ensure that compositionality is maintained as a property of the embeddings for the relationships portrayed in the analogies provided. In other words, equation 2 enforces the following constraint:

"Given a word analogy example a_i in the form of $first - second + third = fourth$, minimize the Hamming Distance between vectors ($first - second + third$) and fourth".

4.3 Evolutionary Operators

To make sure we evolve our candidate embeddings effectively, we define a set of evolutionary oper-

ators with which to generate new chromosomes. For parent selection, we adopt two approaches: 1) *Random* selection and 2) *Roulette Wheel* selection.

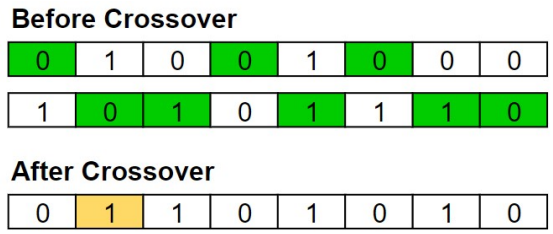

Figure 2: Uniform crossover operation. Green bits are being passed down to the offspring, and yellow bits were mutated after crossover.

Our crossover operation, as shown in Figure 2, takes as input two parent chromosomes (C_1 and C_2) from the population and runs a *uniform* crossover operation, where each parent has an equal probability of contributing any given bit to the resulting offspring. Furthermore, each bit contributed to the offspring has a probability ϵ of mutating (bit-flip), that helps ensure diversity in the population.

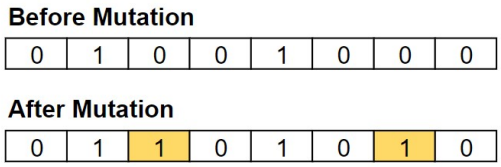

Figure 3: Mutation operation. Yellow bits were flipped as a result of the mutation operation.

In addition to being part of the crossover operator, we also make use of an explicit mutation operation. This operation, as shown in Figure 3, takes in a chromosome, C, and a percentage parameter, δ, from the population as input, returning an offspring with $(\delta * |C|)$ bits flipped.

4.4 Offspring Replacement Strategy

In every generation (iteration) of our algorithm, we maintain a $\mu + \lambda$ replacement strategy; where we generate λ new chromosomes every generation, add them to the population, and then remove the λ worst performers. This results in a consistent population size, no matter how many generations happen, as shown in Figure 4.

Finally, in addition to our previously-outlined genetic operations, we replace one of the non-top-performing chromosomes in our population with a completely random chromosome (after selecting our λ worst performers and removing them) at an interval, γ. This random replacement ensures that as the generations continue, we occasionally see new random solutions inserted that have a chance to help the population escape a local maxima.

5 Experimental Setup

5.1 Dataset

For our experiments, we used Mikolov's word analogy task data-set (Mikolov et al., 2013b), which is comprised of 936 vocabulary words, $8,869$ semantic analogies, and $10,675$ syntactic analogies. For evolutionary pretraining, we split this data-set into five folds and do five-fold cross validation, i.e., we train a binary embedding on four folds' data and test on the remaining fold. In other words, each fold is considered as the testing set once and consists of about $3,900$ unseen analogies from the whole data-set.

5.2 Implementation Details

As part of implementation, we used the following set of parameters: population size (μ) of 25, crossover mutation probability (ϵ) of .01, mutation probability (δ) of .0025, random insertion interval (γ) of 5000, and dimensions (d) of 16, 32, and 64. In each generation, we generated 5 unique offspring: 2 from crossover (one with roulette wheel selection and one with random selection), and 3 mutations (two with roulette wheel selection and one with random selection).

5.3 Evaluation Metrics

For testing, the goal of the word analogy task is to find the fourth word in an analogy of the form "a_1 is to a_2 as a_3 is to a_4". Our evaluation first computes the binary vector $a_1 - a_2 + a_3$ and ranks the closest neighbors by distance. As mentioned before, we use the bitwise *XOR* operation as subtraction and the bitwise *OR* operation as addition.

Using this task, we report the mean reciprocal rank (MRR) as our primary evaluation metric for the generated binary embeddings, along with top-1 and top-5 accuracy scores for each fold. For a given set of analogies, we define MRR as the following:

$$MRR = \frac{1}{|A|} \sum_{i=1}^{|A|} \frac{1}{rank_i} \quad (3)$$

where A is the collection of analogies and $rank_i$ indicates the ordinal position of the correct fourth word in the analogy.

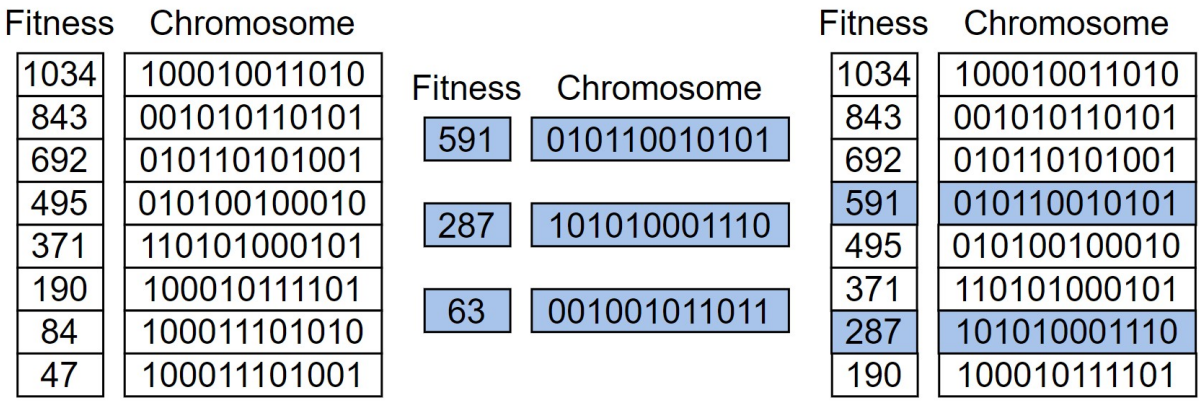

Figure 4: $\mu + \lambda$ selection strategy. The left side indicates the population at the beginning of the current generation, the blue chromosomes indicate newly-generated offspring, and the right side indicates the new population after the λ lowest-performing chromosomes are removed.

6 Results

6.1 Convergence

Figure 5 shows how each size of binary embedding converges as each generation evolves. As depicted, the smaller embeddings converge much faster, with 16-bit embeddings taking roughly 125,000 generations to converge, whereas 64-bit embeddings take more than 400,000 generations to converge.

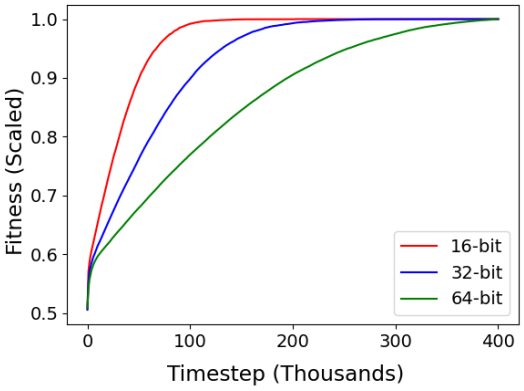

Figure 5: Training convergence for 16, 32, and 64-bit embeddings over 400K generations. Fitness values for each dimension are scaled to $[0, 1]$ for easy comparison.

Furthermore, as shown in Figure 6, the testing performance over time closely mirrors the training convergence. Once again, our 16-bit embedding converges faster than our 64-bit embedding, but it ultimately converges to a lower performance, as shown in Tables 1 and 2, reaching an average MRR of 0.65 whereas the 64-bit embedding reaches an average MRR of 0.68.

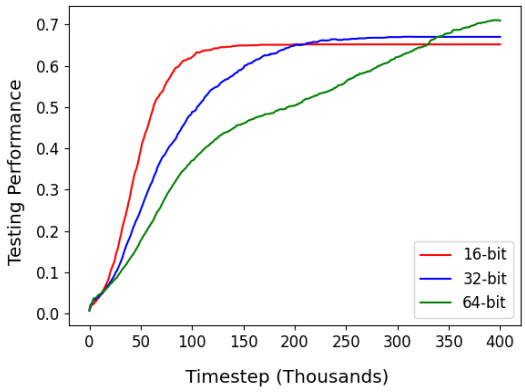

Figure 6: Testing performance for 16, 32, and 64-bit embeddings over 400,000 generations. Performance values for each dimension are scaled to $[0, 1]$ for ease of comparison with the embeddings' training convergence.

6.2 Quantitative Evaluation

Our embeddings' performances are recorded in Table 1, and from this performance, we can make a few observations. First off, 16-bit embeddings work fairly well on this task due to its small vocabulary size. However, this result may not hold up as the vocabulary size scales closer to the 16-bit maximum of 65,536. As the vocabulary scales past that point, we expect that slightly larger embeddings, like 32-bit embeddings, will outperform 16-bit embeddings by a clear margin, a hypothesis we plan to test in our future work.

We also record our embeddings' top-1 accuracy, indicating how well they perform on the analogy task in terms of semantic/syntactic correctness. As shown in Table 2, performance scales similarly to each embedding's MRR performance, which is not surprising. Furthermore, the top-1 accuracy also

Bits	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Avg
16	0.69	0.66	0.63	0.65	0.64	0.65
32	0.66	0.73	0.67	0.67	0.66	0.68
64	0.65	0.69	0.68	0.71	0.66	0.68

Table 1: Mean Reciprocal Rank (MRR) totals for each fold, evaluated against the full analogy set.

Bits	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Avg
16	0.46	0.41	0.35	0.39	0.36	0.39
32	0.38	0.51	0.41	0.41	0.40	0.42
64	0.37	0.44	0.42	0.48	0.39	0.42

Table 2: Percent of analogies where the correct answer is in the top spot. (top-1 accuracy)

Bits	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Avg
16	0.96	0.96	0.96	0.97	0.96	0.96
32	0.98	0.99	0.98	0.98	0.96	0.98
64	0.99	0.99	0.99	0.99	0.99	0.99

Table 3: Percent of analogies where the correct answer is in the top five. (top-5 accuracy)

scales with embedding size, since larger embeddings have more bits to encode information.

As a further indicator of performance, we record our embeddings' top-5 accuracy in Table 3. As demonstrated by the high accuracy numbers ≥ 0.96, no matter what embedding size is used, more closely-related words always make it into the top nearest neighbors for any given word. This clearly demonstrates the validity of the evolved embeddings as well as the feasibility of our proposed evolutionary pretraining approach.

6.3 Qualitative Analysis

To demonstrate the learned relationships in the binary embeddings, we evolved a 32-bit embedding population using the same parameters on the entire analogy set and extracted a few qualitative examples using the final embeddings. In Table 4, we present some sample words, as well as the three nearest words to each one. As shown, the closest-related words always appear in the top three closest neighbors, but we note that due to the specialized nature of the word analogy dataset, words in the vocabulary mainly learn either semantic relationships or syntactic relationships. Nevertheless, this highlights our model's ability to effectively learn and model both semantic and syntactic relationships between vocabulary words.

Overall, the results shown here highlight our model's ability to not only learn the semantic and syntactic relationships between words, but also to maintain the arithmetic properties between the vectors themselves. Due to the nature of the dataset used, the semantic relationships outlined in the analogy task tend to pertain more to geopolitical relationships than other relationships, like synonyms, antonyms, etc. Nevertheless, this still shows our embeddings' effectiveness at learning a targeted vocabulary and relationships based on analogistic reasoning. In future work, we plan on including a way to artificially curate a more general analogy set to train on, so the embeddings learn more general relationships for a larger vocabulary.

quick	predict	japan	california
quicker	predicts	tokyo	anaheim
quickest	predicted	yen	bakersfield
quickly	predicting	japanese	fontana

Table 4: Examples of the closest neighbors to a given word using a 32-bit embedding.

6.4 Training Time

We trained our binary embeddings on an AMD Ryzen Threadripper 3960X running at 2200 MHz, using a single thread for each fold being trained. The base training time for running 200,000 generations is shown in Table 5. We ran our 16-, 32-, and 64-bit embeddings until they reached convergence, and report our results in Section 6.1.

Dimension	Time Taken (HH:MM:SS)
16-bit	49:12:53
32-bit	67:14:10
64-bit	102:42:35

Table 5: Amount of time taken to run 200,000 iterations on a single thread. (Times are in *hh:mm:ss* format)

7 Conclusion

As low-energy computing paradigms like Spiking Neural Networks (SNNs) become increasingly popular for NLP applications, learning accurate binary word embeddings also becomes very important as SNNs can only process binary/spike inputs. At the same time, as backpropagation is tricky in SNNs and simple quantization-based binarization techniques fail to achieve reasonable accuracy, an alternative approach that can learn high-quality binary embeddings has become a pressing need. In this paper, we introduced a new evolutionary approach to

learn binary embeddings from *scratch*, preserving both the semantic/syntactic relationships between words and the arithmetic properties of the embeddings themselves; while bypassing the difficulties associated with implementing backpropagation in SNNs. Experimental results show that the proposed learning technique is both feasible and promising.

8 Limitations

The largest limitation to this work is the dataset used to evolve the population of chromosomes. The word analogy dataset (Mikolov et al., 2013b) has an extremely small vocabulary size, and only includes 2 to 4 words related to each vocabulary word. To address this, we intend to produce a method for creating a large number of "synthetic" word analogies, so that we can provide the intended vocabulary and have the system learn meaningful relationships for all provided words. On the other hand, the bonus to using this type of "restricted" analogy set is that we can use targeted vocabularies for specialized applications at the edge, allowing for even further savings in energy consumption.

Furthermore, our implementation trains these embeddings on a single thread, so our training times are very large. There is *vast* room for improvement with regard to the training time, so we intend on addressing this in future work as well.

Additionally, our genetic algorithm likely still has room left for optimization. As future work, we plan on optimizing the evolution strategy to further cut down the number of generations needed for a given embedding to converge to its top performance.

We also plan to compare this embedding with some other embeddings, both binary and real-valued, to establish our performance with respect to the state-of-the-art. As part of this comparison, we plan to utilize this embedding in some downstream NLP tasks, both in the real-valued domain and in some SNN architectures, to further evaluate its performance.

Acknowledgements

This work is supported by the National Science Foundation under Grant Number CRII-2153394. We thank the four anonymous reviewers for their valuable feedback to help us improve the paper.

References

Daniel Auge, Julian Hille, Etienne Mueller, and Alois Knoll. 2021. A survey of encoding techniques for signal processing in spiking neural networks. *Neural Processing Letters*, 53(6):4693–4710.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Martin V Butz, Kumara Sastry, and David E Goldberg. 2003. Tournament selection: Stable fitness pressure in xcs. In *Genetic and Evolutionary Computation Conference*, pages 1857–1869. Springer.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder. *CoRR*, abs/1803.11175.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364.

Francesco Daghero, Daniele Jahier Pagliari, and Massimo Poncino. 2021. Energy-efficient deep learning inference on edge devices. In *Advances in Computers*, volume 122, pages 247–301. Elsevier.

Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. 2021. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934.

Kalyanmoy Deb. 2011. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*, pages 3–34. Springer.

J Devlin, MW Chang, K Lee, and KB Toutanova. 2019. Pre-training of deep bidirectional transformers for language understanding in: Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers). *Minneapolis, MN: Association for Computational Linguistics*, pages 4171–86.

Anh Viet Do, Mingyu Guo, Aneta Neumann, and Frank Neumann. 2021. Analysis of evolutionary diversity optimisation for permutation problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 574–582.

Wafaa S El-Kassas, Cherif R Salama, Ahmed A Rafea, and Hoda K Mohamed. 2021. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah Smith. 2015. Sparse overcomplete word vector representations. *arXiv preprint arXiv:1506.02004*.

George T Hall, Pietro S Oliveto, and Dirk Sudholt. 2020. Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 823–831.

John H Holland. 1992. Genetic algorithms. *Scientific american*, 267(1):66–73.

Forrest N Iandola, Albert E Shaw, Ravi Krishna, and Kurt W Keutzer. 2020. Squeezebert: What can computer vision teach nlp about efficient neural networks? *arXiv preprint arXiv:2006.11316*.

Murat Ince. 2022. Automatic and intelligent content visualization system based on deep learning and genetic algorithm. *Neural Computing and Applications*, 34(3):2473–2493.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Qiao Jin, Zheng Yuan, Guangzhi Xiong, Qianlan Yu, Huaiyuan Ying, Chuanqi Tan, Mosha Chen, Songfang Huang, Xiaozhong Liu, and Sheng Yu. 2022. Biomedical question answering: A survey of approaches and challenges. *ACM Computing Surveys (CSUR)*, 55(2):1–36.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hervé Jégou, and Tomás Mikolov. 2016. Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Abdullah Ammar Karcioğlu and Ahmet Cahit Yaşa. 2020. Automatic summary extraction in texts using genetic algorithms. In *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE.

Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. 2021. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126.

Youngeun Kim and Priyadarshini Panda. 2021. Optimizing deeper spiking neural networks for dynamic vision sensing. *Neural Networks*, 144:686–698.

Sandeep Kumar, Sanjay Jain, and Harish Sharma. 2018. Genetic algorithms. In *Advances in swarm intelligence for optimizing problems in computer science*, pages 27–52. Chapman and Hall/CRC.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2020. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):50–70.

Yuchen Liang, Chaitanya K. Ryali, Benjamin Hoover, Leopold Grinberg, Saket Navlakha, Mohammed J. Zaki, and Dmitry Krotov. 2021. Can a fruit fly learn word embeddings? *CoRR*, abs/2101.06887.

Huw Lloyd and Martyn Amos. 2017. Analysis of independent roulette selection in parallel ant colony optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 19–26.

Xiaoling Luo, Hong Qu, Yuchen Wang, Zhang Yi, Jilun Zhang, and Malu Zhang. 2022. Supervised learning in multilayer spiking neural networks with spike temporal error backpropagation. *IEEE Transactions on Neural Networks and Learning Systems*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

G Pavai and TV Geetha. 2016. A survey on crossover operators. *ACM Computing Surveys (CSUR)*, 49(4):1–43.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

Chi-Sang Poon and Kuan Zhou. 2011. Neuromorphic silicon neurons and large-scale neural networks: Challenges and opportunities. *Frontiers in Neuroscience*, 5.

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. 2019. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575:607–617.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Catherine D Schuman, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, Bill Kay, et al. 2022. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2(1):10–19.

Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. 2019. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13.

S.N. Sivanandam and S.N. Deepa. 2008. *Genetic Algorithm Optimization Problems*, pages 165–209. Springer Berlin Heidelberg, Berlin, Heidelberg.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2019. Near-lossless binarization of word embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7104–7111.

Aymeric Vie, Alissa M Kleinnijenhuis, and Doyne J Farmer. 2020. Qualities, challenges and future of genetic algorithms: a literature review. *arXiv preprint arXiv:2011.05277*.

Yuxin Wang, Qiang Wang, and Xiaowen Chu. 2020. Energy-efficient inference service of transformer-based deep learning models on gpus. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 323–331. IEEE.

Ashima Yadav and Dinesh Kumar Vishwakarma. 2020. Sentiment analysis using deep learning architectures: a review. *Artificial Intelligence Review*, 53(6):4335–4385.

Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 811–824. IEEE.