Tsetlin Machine Embedding: Representing Words Using Logical Expressions

Bimal Bhattarai and **Ole-Christoffer Granmo** and **Lei Jiao**
Rohan Kumar Yadav and **Jivitesh Sharma**
University of Agder (UiA), Grimstad, Norway
{bimal.bhattarai, ole.granmo, lei.jiao, rohan.yadav, jivitesh.sharma}@uia.no

Abstract

Embedding words in vector space is a fundamental first step in state-of-the-art natural language processing (NLP). Typical NLP solutions employ predefined vector representations to improve generalization by co-locating similar words in vector space. For instance, Word2Vec is a self-supervised predictive model that captures the context of words using a neural network. Similarly, GloVe is a popular unsupervised model incorporating corpus-wide word co-occurrence statistics. Such word embedding has significantly boosted important NLP tasks, including sentiment analysis, document classification, and machine translation. However, the embeddings are dense floating-point vectors, making them expensive to compute and difficult to interpret. In this paper, we instead propose to represent the semantics of words with a few defining words that are related using propositional logic. To produce such logical embeddings, we introduce a Tsetlin Machine-based autoencoder that learns logical clauses self-supervised. The clauses consist of contextual words like "black", "cup", and "hot" to define other words like "coffee", thus being human-understandable. We evaluate our embedding approach on several intrinsic and extrinsic benchmarks, outperforming GloVe on six classification tasks. Furthermore, we investigate the interpretability of our embedding using the logical representations acquired during training. We also visualize word clusters in vector space, demonstrating how our logical embedding co-locate similar words.[1]

1 Introduction

The success of natural language processing (NLP) relies on advances in word, sentence, and document representation. By capturing word semantics and similarities, such representations boost the performance of downstream tasks (Borgeaud et al., 2022), including clustering, topic modelling (Angelov, 2020), searching, and text mining (Huang et al., 2020).

While straightforward, the traditional bag-of-words encoding does not consider the words' position, semantics, and context within a document. Distributed word representation (Bengio et al., 2000; Bojanowski et al., 2017) addresses this lack by encoding words as low-dimensional vectors, referred to as *embeddings*. The purpose is to co-locate similar or contextually relevant words in vector space. There are many algorithms for learning word embeddings. Contemporary self-supervised techniques like Word2Vec (Mikolov et al., 2013), FastText (Bojanowski et al., 2017), and GloVe (Pennington et al., 2014) have demonstrated how to build embeddings from word co-occurrence, utilizing massive training data. Introducing context-dependent embeddings, the more sophisticated language models BERT (Devlin et al., 2019) and ELMO (Peters et al., 2018) now perform remarkably well in downstream tasks (Reimers and Gurevych, 2019). However, they require significant computation power (Schwartz et al., 2020).

The above approaches represent words as dense floating-point vectors. Word2Vec, for instance, typically builds a 300-dimensional vector per word. The size and density of these vectors make them expensive to compute and difficult to interpret. Consider, for example, the word "queen." Representing it with 300 floats seems inefficient compared to the Oxford Language definition for the same word: "the female ruler of an independent state, especially one who inherits the position by right of birth." From this perspective, it appears advantageous to create embeddings directly from words rather than from arbitrary floating-point values. Such interpretable embeddings would capture the multiple meanings of a word using a few defining words,

[1]The Tsetlin Machine Autoencoder and logical word embedding implementation is available here: https://github.com/cair/tmu.

simplifying both computation and interpretation.

In this paper, we propose a Tsetlin Machine (TM) (Granmo, 2018) based autoencoder for creating interpretable embeddings. The autoencoder builds propositional logic expressions with context words that identify each target word. The term “coffee” can, for instance, be represented by “one”, “hot”, “cup”, “table”, and “black”. In this manner, the TM builds contextual representations from a vast text corpus, which model the semantics of each word. In contrast to neural network-based embedding, the logical TM embedding is sparse and energy efficient (Maheshwari et al., 2023; Abeyrathna et al., 2023). The embedding space consists of, e.g., 500 truth values, where each truth value is a logical expression over words. For contextual representation, each target word links to less than ten percent of these expressions. Despite the sparsity and crispness of this representation, it is competitive with neural network-based embedding.

The contributions of our work are summarized below:

- We propose the TM-based Autoencoder to learn efficient encodings in a self-supervised manner. To the best of our knowledge, it is the first logic-based word embedding.
- We introduce TM-based word embedding that builds human-comprehensible contextual representations from unlabeled data.
- We compare our embedding with state-of-the-art approaches on several intrinsic and extrinsic benchmarks, outperforming GloVe on six downstream classification tasks.

2 Related Work

The majority of self-supervised embedding approaches produce dense word representations based on the distributional hypothesis (Harris, 1954), which states that words that occur in the same context are likely to have similar meanings. Word2Vec (Mikolov et al., 2013) is one of the best-known models. It builds embeddings from word co-occurrence using a neural network, leveraging the hidden layer output weights. GloVe (Pennington et al., 2014), on the other hand, embeds by factorizing a word co-occurrence matrix. Similarly, canonical correlation analysis (CCA) is used in (Dhillon et al., 2015) for embedding words to maximize context correlation. In (Levy et al., 2015), it is demonstrated how precise factorization-based SVD can compete with neural embedding. However, all of these methods are challenging to train because they involve tweaking algorithms and hyperparameters toward particular applications (Lample et al., 2016), limiting their wider applicability.

Building upon word embedding, several studies focus on sentence embedding (Arora et al., 2017; Logeswaran and Lee, 2018). Recent advances in sentence embedding include supervised data inference (Reimers and Gurevych, 2019), multitask learning (Cer et al., 2018), contrastive learning (Zhang et al., 2020), and pretrained large language models (Li et al., 2020). However, the majority of sentence embedding techniques overlook intrinsic evaluations, such as similarity tasks, and instead largely focus on extrinsic evaluations involving downstream performance. The most recent building block for embedding originates from the transformer approach (Vaswani et al., 2017). Transformers provide context awareness by utilizing stacks of self-attention layers. BERT (Kenton and Toutanova, 2019), for instance, employs the transformer architecture to carry out extensive self-supervised training, making it capable of producing text embedding. Other embedding models use a contrastive loss function to perform supervised fine-tuning on positive and negative text pairs (Wang et al., 2021). Despite the large variety of text embedding models, they all share three main drawbacks: i) they are computationally demanding to train; ii) they are intrinsically complex because they are trained on a large amount of data to tune a huge amount of parameters; and iii) the embeddings produced from these models are not easily interpreted by humans.

To improve interpretability, Faruqui et al. introduced “Sparse Overcomplete Word Vectors” (SPOWV) which create a sparse non-negative projection of word embedding using dictionary learning (Faruqui et al., 2015). Similarly, SParse Interpretable Neural Embeddings (SPINE) employs a k-sparse denoising autoencoder to generate sparse embeddings (Subramanian et al., 2018). However, these methods are unable to distinguish between multiple context-dependent word meanings. To address this problem, another avenue of research focuses on composing linear combinations of dense vectors from Word2Vec and GloVe (Arora et al., 2018). However, the assumption of linearity does not hold for real-world data, yielding linear coefficients that are difficult to comprehend (Mu et al.,

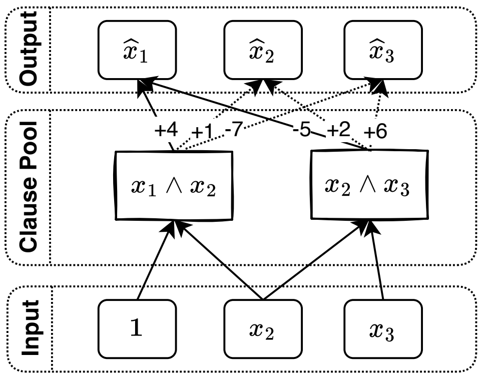

Figure 1: Tsetlin Machine Autoencoder. In this illustration, x_1 is masked by replacing it with value 1 for inferring $\hat{x}_1$.

2017).

The logical embedding approach we present here is most closely related to Naive Bayes word sense induction and topic modeling (Charniak et al., 2013; Lau et al., 2014). This approach learns word meanings from local contexts by considering each instance of the word in a document as a pseudo-document. However, the approach is not scalable because it requires training a single topic per target word. Our approach, on the other hand, is scalable and builds non-linear (non-naive) logical embeddings that capture word compositions. To build the logical embeddings, we propose a novel human-interpretable algorithm based on the TM that provides logical rules describing contexts. The TM has recently performed competitively with other deep learning techniques in many NLP tasks, including novelty detection (Bhattarai et al., 2022a,c), sentiment analysis (Abeyrathna et al., 2023; Yadav et al., 2021), knowledge representation (Bhattarai et al., 2023), and fake news detection (Bhattarai et al., 2022b). Furthermore, the local and global interpretability of TMs have been explored through direct manipulation of the logical rules (Blakely and Granmo, 2021). In addition, TM has been shown to be hardware-friendly for low-power IoT devices (Maheshwari et al., 2023).

3 Tsetlin Machine Autoencoder

We here detail the TM Autoencoder based on the Coalesced TM (Glimsdal and Granmo, 2021), extended with input masking and freezing of masked variables. For ease of explanation, we use three inputs. Adding more inputs follows trivially.

3.1 Architecture

Input and Output. As seen in Figure 1, the TM Autoencoder digests and outputs propositional values: $(x_1, x_2, x_3) \in \{0,1\}^3 \rightarrow (\widehat{x}_1, \widehat{x}_2, \widehat{x}_3) \in \{0,1\}^3$. For our purposes, the propositional variables x_1, x_2, and x_3 each represent a word, for example, "Brilliant", "Actor", and "Awful". The value 1 means that the word occurs in the input text, while the value 0 means that it does not. That is, we represent natural language text as a *set of words*. Notice also that the input variables have corresponding output variables $\widehat{x}_1$, $\widehat{x}_2$, and $\widehat{x}_3$. In short, $\widehat{x}_1$ is to be predicted from x_2 and x_3, $\widehat{x}_2$ from x_1 and x_3, and so on. Continuing our example, $\widehat{x}_1$ predicts the presence of "Brilliant" based on knowing the occurrence of "Actor" and "Awful".

Clause Pool. A pool of n conjunctive clauses, denoted $C_j, j \in \{1, 2, \ldots, n\}$, encodes the input in order to predict the output. A conjunctive clause C_j is simply an *And*-expression over a given subset $L_j \subseteq \{x_1, x_2, x_3\}$ of the input (our autoencoder does not use the input negations $\neg x_1$, $\neg x_2$, and $\neg x_3$):

$$C_j(x_1, x_2, x_3) = \bigwedge_{x_k \in L_j} x_k. \tag{1}$$

For example, the input subset $L_1 = \{x_1, x_2\}$ gives the clause $C_1(x_1, x_2, x_3) = x_1 \wedge x_2$ in the figure. This clause matches the input if x_1 and x_2 both are 1. In our example, the clause accordingly encodes the concept "Brilliant Actor".

Weights. An integer weight matrix $\boldsymbol{W}$ connects each of the n clauses to the three outputs $\widehat{x}_1$, $\widehat{x}_2$, and $\widehat{x}_3$:

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & \cdots & w_{1n} \\ w_{21} & \cdots & w_{2n} \\ w_{31} & \cdots & w_{3n} \end{bmatrix} \in \mathbb{Z}^{3 \times n}. \tag{2}$$

The row index is an output, while the column index is a clause. The weight w_{12}, for instance, connects output $\widehat{x}_1$ to clause C_2. In Figure 1, six weights connect two clauses and three outputs:

$$\begin{bmatrix} +4 & -5 \\ +1 & +2 \\ -7 & +6 \end{bmatrix}. \tag{3}$$

Consider, for example, the weights $(+4, -5)$ of output $\widehat{x}_1$ in the figure. The weight $+4$ states that clause $C_1(x_1, x_2, x_3) = x_1 \wedge x_2$ favours $\widehat{x}_1$ being 1, while clause $C_2(x_1, x_2, x_3) = x_2 \wedge x_3$ opposes it. For example, the concept "Awful Actor" opposes the output "Brilliant".

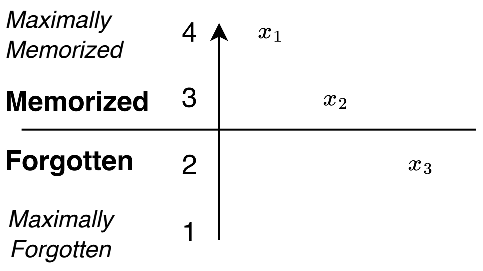

Figure 2: Tsetlin Machine memory for single clause.

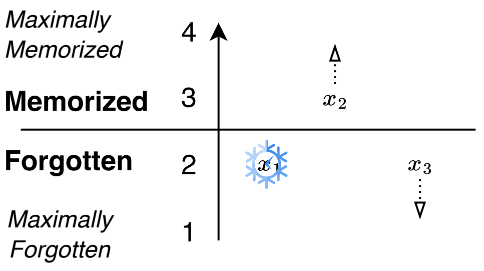

Figure 3: Type Ia (Recognize) Feedback for input $(1, 1, 0)$. The masked variable x_1 is frozen.

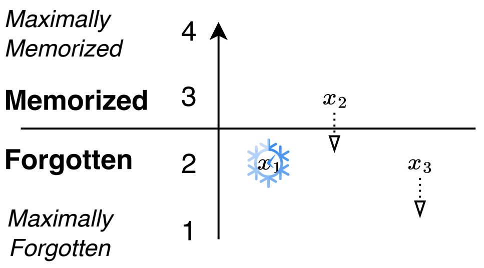

Figure 4: Type Ib (Erase) Feedback for input $(0, 0, 1)$. The masked variable x_1 is frozen.

3.2 Inference

Let us consider the prediction of $\widehat{x}_1$ first. The autoencoder predicts $\widehat{x}_1$ from the clauses and weights:

$$\widehat{x}_1 = 0 \leq \sum_{j=1}^{n} w_{1j} C_j(1, x_2, x_3). \tag{4}$$

That is, each clause C_j is multiplied by its weight w_{1j} for output $\widehat{x}_1$. The outcomes are then summed up to decide the output. If the sum is larger than or equal to zero, the output is $\widehat{x}_1 = 1$. Otherwise, it is $\widehat{x}_1 = 0$. Clauses with positive weight thus promote output $\widehat{x}_1 = 1$ while clauses with negative weight encourage $\widehat{x}_1 = 0$. Notice that x_1 is masked by replacing it with value 1. Accordingly, the autoencoder infers output $\widehat{x}_1$ from the remaining inputs x_2 and x_3.

Correspondingly, $\widehat{x}_2$ and $\widehat{x}_3$ are calculated by respectively masking x_2 and x_3:

$$\widehat{x}_2 \;=\; 0 \leq \sum_{j=1}^{n} w_{2j} C_j(x_1, 1, x_3), \tag{5}$$

$$\widehat{x}_3 \;=\; 0 \leq \sum_{j=1}^{n} w_{3j} C_j(x_1, x_2, 1). \tag{6}$$

Example. Assume that the input is always either $(1, 1, 0)$ or $(0, 1, 1)$. The input $(1, 1, 0)$ could, for instance, represent "Brilliant Actor" and $(0, 1, 1)$ "Awful Actor". Then notice how Eq. (4) correctly determines the masked input x_1 with output $\widehat{x}_1$ in Figure 1, both for input $(1, 1, 0)$ and $(0, 1, 1)$.

3.3 Learning

We next consider how to learn the variable subsets L_j for the clauses $C_j, j \in \{1, 2, \ldots, n\}$, as well as how to determine the weights w_{ij} of the weight matrix W.

Clause Memory. Each clause C_j has a graded memory that contains the input variables, shown in Figure 2. The graded memory enables *incremental* learning of the variable subsets from data. Observe how each variable is in one of four memory positions (the number of memory positions is a user-configurable parameter). Positions $1-2$ mean *Forgotten*. Positions $3-4$ mean *Memorized*. *Memorized* variables take part in the clause, while *Forgotten* ones do not. The memory in Figure 2 thus gives the clause $C_j(x_1, x_2, x_3) = x_1 \land x_2$.

Learning Step. The TM Autoencoder learns incrementally using three kinds of memory and weight updates: Type Ia, Type Ib, and Type II. Each training example has the form $[k, (x_1, x_2, x_3), x_k], 1 \leq k \leq 3$. The first element is an index that identifies which input to mask and which output to predict. The second element is an input vector (x_1, x_2, x_3) and the third element is the target value for output $\widehat{x}_k$, which is x_k. We describe the update procedure step-by-step below for index 1 examples (output $\widehat{x}_1$ prediction). The update procedure for $\widehat{x}_2$ and $\widehat{x}_3$ follows trivially.

Clause Update Probability. First, we calculate the weighted clause sum for $\widehat{x}_1$ from Eqn. (4): $v_1 = \sum_{j=1}^{n} w_{1j} C_j(1, x_2, x_3)$. The sum is then compared with a margin T (hyper-parameter) to calculate a summation error ϵ. The error depends on the x_1-value:

$$\epsilon = \begin{cases} T - \text{clip}(v_1, -T, T), & x_1 = 1, \\ T + \text{clip}(v_1, -T, T), & x_1 = 0. \end{cases} \tag{7}$$

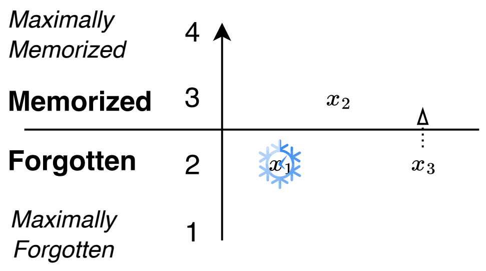

Figure 5: Type II (Reject) Feedback for input $(0, 1, 0)$. The masked variable x_1 is frozen.

That is, for x_1-value 1 the weighted clause sum should become T, while for x_1-value 0 the sum should become $-T$. The goal of the learning is thus to reach the margin for all inputs (x_1, x_2, x_3), ensuring correct output from Equation (4). To reach this goal, each clause C_j is updated randomly with probability $\frac{\epsilon}{2T}$ in each round. In other words, the update probability drops with the error toward zero.

Update Types. The kind of update depends on the values of x_1, $C_j(1, x_2, x_3)$, and w_{1j}. We first consider clauses with positive weight, $w_{1j} \geq 0$. According to Eqn. 4, they are to recognize patterns for $x_1 = 1$. Note that in all of the below updates, the masked variable x_1 is frozen, leaving it unaffected by the update.

- **Type Ia (Recognize) Feedback** occurs when $x_1 = 1$ and $C_j(1, x_2, x_3) = 1$. Then one can say that $C_j(1, x_2, x_3) = 1$ is a *true positive* because it correctly predicts the masked x_1-value. The Type Ia feedback reinforces this successful match by updating the memory of C_j to further mimic the input (see Figure 3). That is, 1-valued variables move one step upwards in memory, with a probability of 1.0.[2] Conversely, 0-valued inputs move one step downwards, however, randomly with probability $\frac{1}{s}$. Here, s is a hyperparameter called *specificity*, meaning that a larger s makes the clauses more specific (Zhang et al., 2022). The clause overall is also reinforced by incrementing its weight w_{j1} by 1.

- **Type Ib (Erase) Feedback** occurs when $x_1 = 1$ and $C_j(1, x_2, x_3) = 0$. Then we call $C_j(1, x_2, x_3) = 0$ a *false negative* because it fails to promote $x_1 = 1$. In that case, all inputs randomly move one step downward in memory (see Figure 4). Again, each downward move happens with probability $\frac{1}{s}$. Here, the purpose is to eliminate the false negative outcome by erasing variables from the clause.

- **Type II (Reject) Feedback** occurs when $x_1 = 0$ and $C_j(1, x_2, x_3) = 1$. Then, one can say that $C_j(1, x_2) = 1$ is a *false positive* because it promotes $x_1 = 1$ when in fact we have $x_1 = 0$. Then all *Forgotten* 0-valued inputs move one step upwards in memory. The purpose is to eventually eliminate the current false positive outcome by injecting 0-valued variables into the clause. The clause is further diminished by decrementing its weight w_{1j} by 1. Note that the latter decrement can switch the weight from positive to negative. In effect, the clause then changes role, now training to recognize $x_1 = 0$ instead.

Clauses C_j with negative weights, $w_{1j} < 0$, are updated the same way. However, they are to recognize patterns for $x_1 = 0$. To achieve this, $x_1 = 0$ is treated as $x_1 = 1$ and $x_1 = 1$ is treated as $x_1 = 0$ when updating the memories. Furthermore, the weight updates are reversed. Increments become decrements, and vice versa.

Algorithm 1 TM word embedding

```
Require: Vocabulary $\mathcal{V}$; Documents $\mathcal{D} \in \mathcal{G}, \mathcal{D} \subseteq \mathcal{V}$; Accumulation $u$; Clauses $n$; Margin $T$; Specificity $s$; Rounds $r$
1:  TMCreate$(n, T, s)$    ▷ Create TM with $n$ clauses.
2:  for $r$ rounds do
3:      for $word_k \in \mathcal{V}$ do    ▷ Create one example per word.
4:          $q_k \leftarrow$ Select$(\{0, 1\})$    ▷ Random target value.
5:          if $q_k = 1$ then
6:              $\mathcal{G}_k \leftarrow \{\mathcal{D} | word_k \in \mathcal{D}, \mathcal{D} \in \mathcal{G}\}$    ▷ Documents with $word_k$.
7:          else
8:              $\mathcal{G}_k \leftarrow \{\mathcal{D} | word_k \notin \mathcal{D}, \mathcal{D} \in \mathcal{G}\}$    ▷ Documents without $word_k$.
9:          $\mathcal{S}_k \leftarrow$ SelectN$(\mathcal{G}_k, u)$    ▷ Random subset of size $u$.
10:         $\mathcal{U}_k \leftarrow \bigcup_{\mathcal{D} \in \mathcal{S}_k} \mathcal{D}$    ▷ Union of selected documents.
11:         $\boldsymbol{x}_k \leftarrow (x_1, x_2, \ldots, x_m), x_i = \begin{cases} 1, & word_i \in \mathcal{U}^k \\ 0, & word_i \notin \mathcal{U}^k \end{cases}$
12:         TMUpdate$(k, \boldsymbol{x}_k, q_k)$    ▷ Update TM Autoencoder for output index $k$, input $\boldsymbol{x}_k$, and target value $\widehat{x}_k = q_k$.
13: $\mathcal{C}, \boldsymbol{W} \leftarrow$ TMGetState()    ▷ Clauses $C_j \in \mathcal{C}$ with weights $\boldsymbol{W}$.
14: $\boldsymbol{E} \leftarrow$ clip$(\boldsymbol{W}, 0, T)$    ▷ Elementwise clip of negative values produces weighted logical word embeddings.
15: $\boldsymbol{B} \leftarrow (\boldsymbol{W} > 0)$    ▷ Elementwise comparison with zero produces purely logical word embeddings.
16: return $\mathcal{C}, \boldsymbol{E}, \boldsymbol{B}$
```

[2]Originally, the increment probability is $\frac{s-1}{s}$, which can be boosted to 1.0 to enhance the learning of true positive patterns (Granmo, 2018).

Dataset	W2V			FastText			TM			GloVe		
	Spearman	Kendall	Cosine	Spearman	Kendall	Cosine	Spearman	Kendall	Cosine	Spearman	Kendall	Cosine
WordSim-353	0.53	0.37	0.87	0.46	0.32	0.79	0.45	0.31	0.90	0.41	0.28	0.90
SimLex-999	0.26	0.18	0.79	0.23	0.16	0.79	0.14	0.10	0.76	0.25	0.17	0.80
MEN	0.71	0.50	0.91	0.71	0.51	0.94	0.64	0.45	0.94	0.73	0.53	0.95
MTurk-287	0.66	0.47	0.77	0.63	0.44	0.93	0.63	0.44	0.92	0.66	0.47	0.86
MTurk-771	0.57	0.39	0.86	0.52	0.36	0.93	0.48	0.32	0.91	0.58	0.40	0.94
RG-65	0.72	0.58	0.89	0.67	0.49	0.88	0.75	0.63	0.92	0.78	0.62	0.93
Average	0.58	0.42	0.85	0.54	0.38	0.88	0.52	0.38	0.89	0.57	0.42	0.90

Table 1: Performance comparison of TM embedding with baseline algorithms on the similarity task.

4 Logical Embedding Procedure

We now use the TM Autoencoder to build logical embeddings. Let $\mathcal{V} = \{word_1, word_2, \ldots, word_m\}$ be the target vocabulary consisting of m unique words.

Pre-processing. The first step is to pre-process the document corpus. To this end, each document is represented by a subset of words $\mathcal{D} \subseteq \mathcal{V}$. For example, the document "The actor was brilliant" becomes the set $\mathcal{D} = \{$"actor", "brilliant", "the", "was"$\}$. The set $\mathcal{G}$, in turn, contains all the documents, $\mathcal{D} \in \mathcal{G}$. Finally, in propositional vector form, the word set $\mathcal{D}$ becomes:

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_t), x_i = \begin{cases} 1, & word_i \in D, \\ 0, & word_i \notin D. \end{cases} \tag{8}$$

Embedding. Algorithm 1 specifies the procedure for embedding the m vocabulary words from $\mathcal{V}$ by using n clauses, $C_j, 1 \leq j \leq n$, forming a clause set $\mathcal{C}$. Each round of training produces a training example $[k, (x_1, x_2, \ldots, x_m), q_k]$ per $word_k$ in $\mathcal{V}$. First, a target value q_k for the word is set randomly to either 0 or 1. This random selection balances the dataset. If q_k becomes 1, we randomly select u documents that contain $word_k$ and assign them to the set $\mathcal{S}_k$ (positive examples). Otherwise, we randomly select u documents that do *not* contain the word (negative examples). Next, the randomly selected documents are merged by ORing them together, yielding the unified document $\mathcal{U}_k$. The purpose of ORing multiple documents is to increase the frequency of rare context words. Then, picking up characteristic ones becomes easier. After that, the propositional vector form $(x_1, x_2, \ldots, x_m)$ of $\mathcal{U}_k$ is obtained. Finally, the TM Autoencoder is updated with $[k, (x_1, x_2, \ldots, x_m), q_k]$ following the training procedure in Section 3.

Vector Space Representation. The weighted logical embedding of $word_k \in \mathcal{V}$ can now be obtained from row k of a matrix $\boldsymbol{E}$ (returned from Algorithm 1), while the purely logical embedding is found in row k of the matrix $\boldsymbol{B}$. Let $\boldsymbol{e}_k$ denote the k'th row of $\boldsymbol{E}$, and let $\boldsymbol{e}_l$ denote the l'th row. We can then compare the similarity of two words $word_k$ and $word_l$ using cosine similarity (CS) between their $\boldsymbol{E}$-embedding:

$$CS(word_k, word_l) = \frac{\boldsymbol{e}_k \cdot \boldsymbol{e}_l}{||\boldsymbol{e}_k||\ ||\boldsymbol{e}_l||}. \tag{9}$$

Dataset	W2V	FastText	TM	GloVe
AP	0.50	0.35	0.41	0.41
BLESS	0.64	0.66	0.62	0.66
ESSLI-2008	0.63	0.60	0.57	0.56
Average	0.59	0.54	0.53	0.54

Table 2: Performance comparison of TM embedding with baseline embeddings on the categorization task.

5 Empirical Evaluation

We here evaluate our logical embedding scheme, comparing it with neural network approaches.

Datasets and Setup We first evaluate our logical embedding intrinsically, followed by an extrinsic evaluation using classification tasks.

Intrinsic Evaluation. We use word similarity and categorization benchmarks for intrinsic evaluation. That is, we examine to what degree our approach retains semantic word relations. To this end, we measure how semantic relations manifest in vector space using six datasets: SimLex-999, WordSim-353, MEN, MTurk-287, MTurk-771, and RG-65. Each dataset consists of human-scored word pairs, which are compared with the corresponding vector space similarities. The categorization tasks evaluate how well we can group words into distinct word categories only based on their embedding. We here use three datasets: AP, BLESS, and ESSLLI-2008. To obtain the categorization accuracy, we use KMeans clustering from sklearn on the word embeddings and examine the cluster quality by calculating the purity score from (https://github.com/purity). As baselines, we chose Word2Vec, GloVe, and FastText because of their wide use.

Extrinsic Evaluation. In our extrinsic evaluation, we investigate how well our logical embedding supports downstream NLP classification tasks. Using the word embeddings as feature vectors, the performance of supervised classification models gives insight into the embedding quality. We employ six standard text classification datasets from SentEval (Conneau and Kiela, 2018): R8, R52, TREC, SUBJ, SST-2, and SST-5. For supervised learning, we use the standard attention-based BiLSTM model with the Adam optimizer and cross-entropy loss function. In this manner, we directly contrast GloVe embedding against the logical TM approach.

Embedding Datasets. For extrinsic evaluation with BiLSTM, we use standard 300-dimensional GloVe embeddings, pre-trained on the *Wikipedia 2014 + Gigaword 5* datasets (6B tokens).[3] The purpose is to compare the TM embedding performance against widely used and successful GloVe embeddings on downstream tasks. To directly compare the intrinsic properties of Word2Vec, GloVe, FastText, and TM embedding, we also train them from scratch using the One Billion Word dataset (Chelba et al., 2014). For training the TM, we use $r = 2000$ training rounds, producing 2000 examples per word by accumulating $u = 25$ contexts per example. We use the following hyperparameters: a pool of $n = 600$ clauses, margin $T = 1200$, and specificity $s = 5.0$.[4] Word2Vec Skip-Gram is trained with 10 passes over the data, using separated embeddings for the input and output contexts. The window size is 5 and we use five negative samples per example. Similarly, GloVe is trained for 30 epochs with a window size of 10 and a learning rate of 0.05. While Word2Vec and FastText have been trained using the standard gensim library (https://github.com/gensim/), GloVe has been trained using https://github.com/maciejkula/glove-python.

5.1 Results and Discussion

As presented in Section 5, we employ two kinds of evaluation: intrinsic and extrinsic. Table 1 contains the intrinsic evaluation results from six word similarity tasks. We here compute the Spearman correlation, the Kendall coefficient, and the cosine similarity between the human-set similarity scores and the predicted similarity scores per dataset. Considering Spearman and Kendall scores, Word2Vec and GloVe are marginally better than the comparable FastText and TM embedding. However, as reported in (Rastogi et al., 2015), small differences in correlation-based measures are not necessarily significant for smaller datasets. To more robustly assess performance, we therefore also use cosine similarity to compare predicted word similarities with the human-set similarities. In terms of cosine score, our model outperforms Word2Vec and FastText on the majority of the datasets, while performing competitively with GloVe. This means that the angles between the human-set similarities and the GloVe/TM-predicted similarities are quite similar. Finally, Table 2 shows the outcome for the word categorization tasks. As seen, the performance of the selected embedding techniques are comparable, with Word2Vec being slightly ahead.

Previous research indicates that intrinsic word similarity performance is minimally or even negatively correlated with downstream NLP performance (Wang et al., 2021). Therefore, we also include an extrinsic evaluation with six downstream classification tasks. To avoid overfitting and robustly assess downstream properties, we keep our experimental setup as above. Table 3 reports the outcome of the evaluation, where the embeddings have been fed to an attention-based BiLSTM model. The first configuration (GloVe) uses the pre-trained GloVe embeddings from the *Wikipedia 2014 + Gigaword 5* datasets. The second configuration consists of our purely logical TM embedding from One Billion Word (embedding $\boldsymbol{B}$ from Algorithm 1). Being five times smaller, the One Billion Word dataset only provides about 80 percent of the vo-

Dataset	GloVe		TM		$\text{TM}_{\text{hybrid}}$	
	Acc.	F1	Acc.	F1	Acc.	F1
R8	96.31	0.88	96.10	0.88	97.80	0.94
TREC	95.20	0.95	96.40	0.96	96.80	0.96
R52	90.34	0.58	91.23	0.62	94.23	0.68
SUBJ	86.20	0.86	85.80	0.85	86.70	0.87
SST-2	76.38	0.75	75.61	0.74	79.30	0.78
SST-5	47.47	0.46	47.80	0.43	49.75	0.44

Table 3: Performance comparison of our embedding with standard GloVe embedding on the classification task.

[3]The pre-trained GloVe embeddings can be found here: https://nlp.stanford.edu/projects/glove/

[4]The TM Autoencoder and logical word embedding implementation can be found here: https://github.com/cair/tmu.

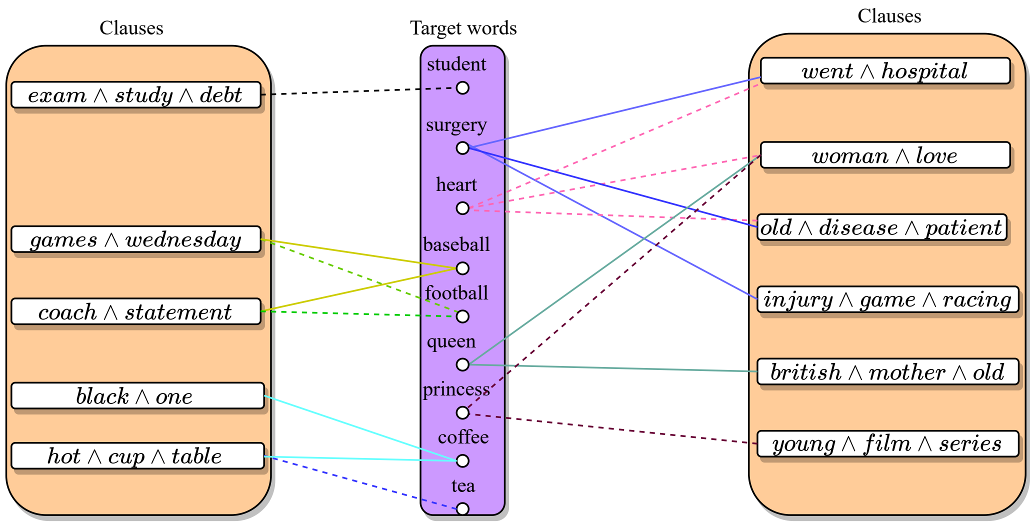

Figure 6: Interpretability of clauses capturing distinct meanings of target words in the TM embedding.

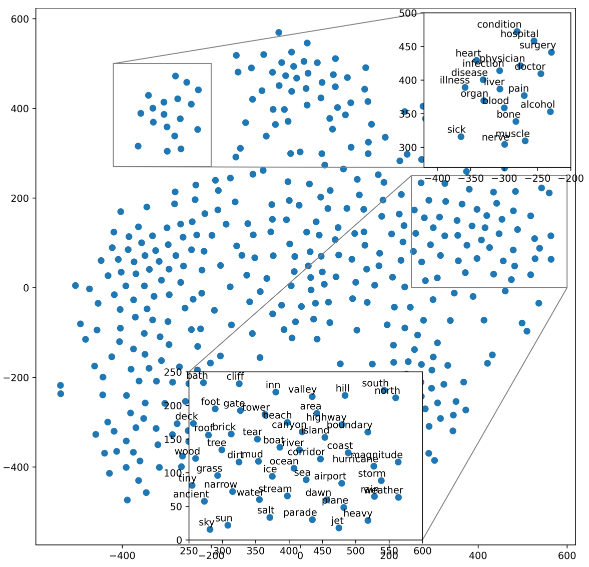

Figure 7: TM embedding visualization plotted using t-SNE.

cabulary required for the classification tasks. We embed the remaining 20 percent of the words randomly. Hence, the TM approach can potentially have a disadvantage in the evaluation. In the third configuration (TM$_{\text{hybrid}}$), we replace the 20 percent random embeddings with the corresponding GloVe embeddings (approximately 80% TM + 20% GloVe). We note that the downstream accuracy of BiLSTM is similar for both TM and GloVe. Specifically, the TM embedding exceeds GloVe by a small margin on TREC, R52, and SST-5. The hybrid embedding, on the other hand, clearly outperforms the other two. In particular, for R52, SST-2, and SST-5, the hybrid embedding is able to surpass GloVe by a substantial margin of roughly $2-4\%$. Given that the datasets are not completely balanced, we also compute F1 macro scores. We again observe that the TM embedding either outperforms or is competitive with GloVe. For R8 and R52, the hybrid embedding surpasses GloVe by a large margin, respectively, by around 6% and 10%. Based on these results, we conjecture that logical TM embedding can successfully replace neural network embedding. Even with 20% of the vocabulary missing, trained on five times smaller data, the logical embedding performs competitively with GloVe. Interestingly, the hybrid approach performed even better. One possible explanation for this higher performance could be the extra information added by the larger vocabulary. Additionally, there may be synergy between the neural and logical representations that manifest in the hybrid approach.

5.2 Interpretability and Visualization

In this section, we investigate the nature of the TM embeddings in more detail, focusing on interpretability. Our embedding consists of the positive clause weights $\boldsymbol{E}$, or, alternatively, the propositional version $\boldsymbol{B}$, explained by the set of clauses $\mathcal{C}$. As demonstrated in Figure 6, each clause in $\mathcal{C}$ captures a facet of a context. The dotted lines in the figure showcase the connection between the target words and their clauses from matrix $\boldsymbol{B}$ (and, accordingly, $\boldsymbol{E}$). Each target word gets its own color to more easily discern the connections. In the figure, we provide an excerpt of 18 connections from $\boldsymbol{B}$, involving 9 target words and the 11 most triggered clauses for these words. Consider, for example, the target words *surgery* and *heart*. These two target words share two clauses: [$went \wedge hospital$]

and [$old \wedge disease \wedge patient$]. The two clauses capture two joint contexts, both related to health. The clauses thus represent commonality between the target words, providing information on one particular meaning of the words.

The two target words are also semantically different. The differences are captured by the clauses they do not share. The target word *heart*, for example, also relates to the meaning [$woman \wedge love$], which *surgery* does not. *Surgery*, on the other hand, connects with [$injury \wedge game \wedge racing$]. In this manner, the unique meanings and relations between words are represented through the sharing of logical expressions. Accordingly, it is feasible to capture a wide range of possible contextual representations with concise logical expressions. As such, the logical embedding provides a sparse representation of words and their relations. Indeed, at most 10% of the clauses connect to each word in our experiments. As shown in the intrinsic evaluations from the previous subsection, these contextual representations are effective for measuring word similarity and categorizing words. Similarly, we observed that the logical embedding is boosting downstream NLP classification tasks.

To cast further light on the TM embedding approach, we visualize the embedding of 400 words from the SimLex-999 dataset in Figure 7, plotted using t-SNE. The figure indicates that we are able to cluster contextually similar words in vector space. To scrutinize the clusters, we zoom in on two of them. Consider the upper-right cluster first. Notice how the words in the cluster relate to *hospital*, such as *heart* and *diseases*. As seen, the word embeddings are closely located in vector space. Similarly, we can observe that terminology connected to weather and geography are grouped together in the bottom cluster. From these two examples, it seems clear that the TM embedding incorporates semantic relationships among words.

6 Conclusion and Future Work

In this work, we first discussed the challenge and necessity of finding computationally simpler and more interpretable word embedding approaches. We then motivated an efficient self-supervised approach, namely, a TM-based autoencoder, for producing sparse and interpretable logical word embeddings. We evaluated our approach on a wide range of intrinsic and extrinsic tasks, demonstrating that it is competitive with dense neural network-based embedding schemes such as Word2Vec, GloVe, and FastText. Further, we investigated the interpretability of our embedding through visualization and a case study. Our conclusion from the study is that logical embedding is able to represent words with logical expressions. This structure makes the representation sparse, enabling a clear-cut decomposition of each word into sets of semantic concepts. Future work includes scaling up our implementation using GPUs to support the building of large-scale vocabularies from more massive datasets.

7 Limitations

The primary purpose of the experiments conducted in the context of the downstream classification task is to thoroughly analyze and comprehend the practical implementation of our embedding approach. Consequently, the evaluation did not involve a comparison of performance against other contemporary transformer-based large language models, such as BERT, which are considered the state of the art. Further, we intend to investigate how sentence-level and document-level embedding can be created using clauses, for instance, applicable for downstream sentence similarity tasks.

References

K. Darshana Abeyrathna, Ahmed Abdulrahem Othman Abouzeid, Bimal Bhattarai, Charul Giri, Sondre Glimsdal, Ole-Christoffer Granmo, Lei Jiao, Rupsa Saha, Jivitesh Sharma, Svein Anders Tunheim, and Xuan Zhang. 2023. Building concise logical patterns by constraining tsetlin machine clause size. In *International joint conference on artificial intelligence (IJCAI)*.

Dimo Angelov. 2020. Top2vec: Distributed representations of topics. *arXiv preprint arXiv:2008.09470*.

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. 2018. Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of ICLR*.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. *Advances in neural information processing systems*, 13.

Bimal Bhattarai, Ole-Christoffer Granmo, and Lei Jiao. 2022a. A Tsetlin Machine Framework for Universal

Outlier and Novelty Detection. In *International Conference on Agents and Artificial Intelligence*, pages 250–268. Springer.

Bimal Bhattarai, Ole-Christoffer Granmo, and Lei Jiao. 2022b. Explainable tsetlin machine framework for fake news detection with credibility score assessment. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 4894–4903.

Bimal Bhattarai, Ole-Christoffer Granmo, and Lei Jiao. 2022c. Word-level human interpretable scoring mechanism for novel text detection using tsetlin machines. *Applied Intelligence*.

Bimal Bhattarai, Ole-Christoffer Granmo, and Lei Jiao. 2023. An interpretable knowledge representation framework for natural language processing with cross-domain application. In *Advances in Information Retrieval: 45th European Conference on Information Retrieval, ECIR 2023, Dublin, Ireland, April 2–6, 2023, Proceedings, Part I*, pages 167–181.

Christian D. Blakely and Ole-Christoffer Granmo. 2021. Closed-Form Expressions for Global and Local Interpretation of Tsetlin Machines. In *34th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2021)*. Springer.

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.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *Proceedings of ICML*, pages 2206–2240. PMLR.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder for English. In *Proceedings of EMNLP*, pages 169–174.

Eugene Charniak et al. 2013. Naive Bayes word sense induction. In *Proceedings of EMNLP*, pages 1433–1437.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, T. Brants, Phillip Todd Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. In *Interspeech*.

Alexis Conneau and Douwe Kiela. 2018. SentEval: An Evaluation Toolkit for Universal Sentence Representations. In *Proceedings of LREC*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*.

Paramveer S Dhillon, Dean P Foster, and Lyle H Ungar. 2015. Eigenwords: spectral word embeddings. *J. Mach. Learn. Res.*, 16:3035–3078.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A Smith. 2015. Sparse Overcomplete Word Vector Representations. In *Proceedings of ACL, (Long Papers)*, pages 1491–1500.

Sondre Glimsdal and Ole-Christoffer Granmo. 2021. Coalesced Multi-Output Tsetlin Machines with Clause Sharing. *Available as an arXiv preprint, arXiv:2108.07594*.

Ole-Christoffer Granmo. 2018. The tsetlin machine - a game theoretic bandit driven approach to optimal pattern recognition with propositional logic. *arXiv preprint arXiv:1804.01508*.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in Facebook search. In *Proceedings of ACM SIGKDD*, pages 2553–2561.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In *Proceedings of NAACL-HLT*, pages 260–270.

Jey Han Lau, Paul Cook, Diana McCarthy, Spandana Gella, and Timothy Baldwin. 2014. Learning word sense distributions, detecting unattested senses and identifying novel senses using topic models. In *Proceedings of ACL, (Long Papers)*, pages 259–270.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the association for computational linguistics*, 3:211–225.

Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. 2020. On the Sentence Embeddings from Pre-trained Language Models. In *Proceedings of EMNLP*, pages 9119–9130.

Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. In *Proceedings of ICLR*.

Sidharth Maheshwari, Tousif Rahman, Alex Yakovlev, Ashur Rafiev, Lei Jiao, Ole-Christoffer Granmo, et al. 2023. REDRESS: Generating compressed models for edge inference using Tsetlin machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of ICLR*.

Jiaqi Mu, Suma Bhat, and Pramod Viswanath. 2017. Geometry of polysemy. In *Proceedings of ICLR*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of 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 *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2227–2237.

Pushpendre Rastogi, Benjamin Van Durme, and Raman Arora. 2015. Multiview LSA: Representation learning via generalized CCA. In *Proceedings of NAACL*, pages 556–566.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of EMNLP-IJCNLP*, pages 3982–3992.

Roy Schwartz, Jesse Dodge, Noah Smith, and Oren Etzioni. 2020. Green AI. *Communications of the ACM*, 63:54 – 63.

Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard Hovy. 2018. Spine: Sparse interpretable neural embeddings. In *Proceedings of AAAI*, volume 32.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Kexin Wang, Nils Reimers, and Iryna Gurevych. 2021. Tsdae: Using transformer-based sequential denoising auto-encoderfor unsupervised sentence embedding learning. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 671–688.

Rohan Yadav, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. 2021. Human-Level Interpretable Learning for Aspect-Based Sentiment Analysis. In *Proceedings of AAAI*.

Xuan Zhang, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. 2022. On the Convergence of Tsetlin Machines for the IDENTITY-and NOT Operators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6345–6359.

Yan Zhang, Ruidan He, Zuozhu Liu, Kwan Hui Lim, and Lidong Bing. 2020. An Unsupervised Sentence Embedding Method by Mutual Information Maximization. In *Proceedings of EMNLP*, pages 1601–1610.