Learning Binary Representations End-to-end for Sentence Semantic Matching

Zhuofei Ding

Master of Science
Cognitive Science
School of Informatics
University of Edinburgh
2021

Abstract

Sentence semantic matching is a fundamental task in natural language processing (NLP), aiming at using models to predict semantic relevance between sentences. In real-world applications, sentences are usually pre-computed as continuous representations and stored in memory to reduce inference latency. However, storing continuous embeddings requires a large memory footprint, which leads to engineering challenges. A promising way to alleviate this problem is using binarized embeddings to represent sentences.

This thesis evaluates several existing binarization methods on matching performance, memory footprint and inference latency, using continuous representations produced by baselines on two datasets. We propose a novel architecture BinSiamese to learn binary representations end-to-end. BinSiamese achieves the state-of-the-art, resulting in 16x compression for a 6.6% loss in performance on one dataset, and 4x compression with a 3% improvement in performance on the other dataset. We perform an ablation study to understand the effects of the individual components on BinSiamese, as well as the representation dimension.

Acknowledgements

I would like to thank my supervisors, Cristina Morariu, Viraat Aryabumi and Ben Allison, for their continuous support and invaluable suggestions. I would also thank my best friend Rio, who gives me countless help during my self-isolation. Finally, I would like to thank my family and my girlfriend for their unconditional support and love.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Zhuofei Ding*)

Table of Contents

Chapter 1

Introduction

Sentence semantic matching is a fundamental technology in natural language processing (NLP) [1], aiming at using models to identify the relatedness of sentences [2]. It is the core component of many NLP systems such as Question Answering (Q&A) system [3], Information Retrieval (IR) system [4] and Paraphrase Identification (PI) system [5]. The recent development of deep learning has spawned many NLP models [6], pushing the sentence semantic matching performance to a higher level. A common deep learning architecture for sentence matching is Siamese Network [7], which uses encoders to produce deep representations for sentences, then feeds representations to a downstream component such as distance metrics calculator and Softmax classifier to calculate the degree of matching. A lot of research has remarkably contributed to improving the quality of sentence embeddings [8, 9, 10], but most neglect the potential problem in real-world applications.

In a real-world system that involves the sentence semantic matching component, there is usually a matching database that contains candidate sentences. The candidate sentences are pre-computed to real-value sentence embeddings by the encoder. The system only needs to compute the representation for the query sentence and search for the matching results in the database to reduce the online inference latency. This requires large storage or memory footprint [11], which leads to an extra expenditure to update the hardware to deploy the application. Besides, calculating the similarity between real-value representation vectors is computationally expensive [11]. Thus it can be time-consuming when doing retrieval. This problem is more evident in low-resource devices [12]

Binarizing the real-value sentence representations can mitigate the above issues. Continuous sentence embeddings are compressed as binary form, significantly reducing the memory footprint. The Hamming distance can measure the semantic relevance between binary representations, whose computation cost is less than cosine similarity or dot product. Besides, binary representations are also demonstrated to be faster on GPU at run-time [13]. Recent work in binarized sentence representations [11, 12] has several limitations: *i*) they only used the general-purpose InferSent [8] as the testbed sentence embeddings to evaluate binarization approaches, while different approaches may perform differently using different sentence embeddings; *ii*) the evaluation of binarization approaches ignored the latency of generating representations for the new coming sentences; *iii*) they only considered binarizing the pre-trained sentence embedding vectors rather than training the binary representation end-to-end, while Chen et al. [14] claimed that learning discrete representation in an end-to-end manner could lead to better performance[1].

This thesis focuses on addressing the previous work's limitations. We evaluate several binarization methods proposed in [11] and [12] on two types of sentence semantic matching task (regression and classification). The evaluation perspectives we concern include matching performance, memory footprint and inference latency. We define the inference latency as the sum of two parts, one is the time needed to produce the query sentence embedding, and the other is the time needed to compute the matching scores, while previous work only considered the second part. MaLSTM [15], and SBERT [9] are selected to produce continuous sentence embeddings as testbeds, making up the limitation of the single testbed. We observe different methods perform variously on downstream tasks when using different continuous representations. Thus, intensive experiments need to do to figure out the best binarization approach for a specific task, which is costly.

In this regard, we hypothesize that end-to-end training can provide guidance to produce high-quality binary representations that are adoptive to specific tasks. Motivated by semantic-preserving loss [11], we propose an end-to-end Siamese architecture BinSiamese to generate high-quality binary representations. The BinSiamese is flexible to use different pre-trained encoders. We design two models called BinLSTM and BinSBERT by applying LSTM and BERT as the encoder. On SICK [16] and MRPC [17] datasets, our proposed models achieve state-of-the-art matching performance. BinSBERT even outperforms the SBERT on the MRPC dataset. Besides, we conduct the nearest neighbour retrieval experiment on the SICK dataset and intuitively demonstrate our binary representations can reflect the semantic meaning of sentences.

[1]This argument is also mentioned in the Student's Informatics Proposal.

Moreover, we perform the ablation study to explore the sensitivity of matching performance to the representation dimension and the effect of semantic-preserving loss. We observe that the sensitivities of various methods are different on different types of tasks, and semantic-preserving loss may lead to fluctuation during training. Based on our observations, we summarize some suggestions for choosing suitable binarization approaches and the reasonable coefficient of semantic-preserving loss. We also discuss the possible solution to mitigate the fluctuation caused by semantic-preserving loss.

1.1 Contributions

The contributions of this thesis are summarized as follows:

i) we evaluate several existing binarization approaches based on two different types of continuous representations. The results demonstrate that the matching performance of binarization approaches is related to the different continuous representations. Besides, a more reasonable method is designed to measure the answer latency of different approaches.

ii) a state-of-the-art end-to-end Siamese architecture is proposed to produce high-quality binary representations, demonstrating end-to-end training can bring improvement in matching performance. The architecture also provides a new research idea to obtain binary sentence representations for the academic.

iii) some suggestions for choosing binarization approaches and reasonable coefficient of semantic-preserving loss are summarized based on the ablation study.

1.2 Outline

The remainder of this dissertation is structured as follows:

- **Chapter 2** introduces the necessary knowledge of sentence semantic matching and representation binarization, and also reviews related work.
- **Chapter 3** introduces our comparison framework for binarization approaches.
- **Chapter 4** describes our proposed BinSiamese architecture and also two available encoders.
- **Chapter 5** presents our experiment setup and analyses the experiment results.
- **Chapter 6** concludes our findings and proposes suggestions for future work.

Chapter 2

Background & Related Work

This chapter introduces the necessary background knowledge of the sentence semantic matching task and commonly used deep learning models and reviews the related work to binarization approaches[1].

2.1 Sentence Semantic Matching

Sentence semantic matching exists in many NLP tasks. In the Question Answering task, sentence semantic matching is designed to predict the degree of matching between the query and preset questions or the question and the preset answers [3]. In Information Retrieval, sentence semantic matching is required to measure the similarity between the query and the keys to return the most related values [4]. In the Paraphrase Identification task, sentence semantic matching is used to determine whether two sentences have the same meaning or not [5].

Sentence semantic matching task can be described as finding a mapping to predict the semantic relevance between sentences. Given a sentence $s_i = \{x_1, x_2, ..., x_m\}$, where x_i is a word or character. Define the training set $\mathbb{D} = \{(s_i^{(k)}, s_j^{(k)}, y^{(k)})\}_{k=1}^{N}$, where $y^{(k)}$ is the matching score reflecting the semantic relatedness between $s_i^{(k)}, s_j^{(k)}$. The sentence semantic matching task is to find a function $f(s_i^{(k)}, s_j^{(k)})$ based on $\mathbb{D}$ to predict the matching score $y^{(k)}$. Note that if $y^{(k)} \in \mathbb{R}$, the task is a regression task. If $y^{(k)}$ is categorical variable (e.g. $y^{(k)} \in \{0, 1\}$) the task is a classification task.

The evaluation methods are different depending on the task type. For regression task, Pearson correlation (τ) and Spearman correlation (ρ) are usually used to measure

[1]This chapter is constructed mainly based on the student's Informatics Project Proposal and Informatics research review.

the correlation between the predicted score ($\hat{y}$) and the true score (y).

$$\tau = \frac{\mathrm{cov}(y,\hat{y})}{\sigma_y \sigma_{\hat{y}}} \tag{2.1}$$

$$\rho = \frac{\mathrm{cov}(rg_y, rg_{\hat{y}})}{\sigma_{rg_y} \sigma_{rg_{\hat{y}}}} \tag{2.2}$$

Where cov is the covariance, σ is the standard deviation and rg_y is the rank of y. If the task is classification, the accuracy (acc) is usually used to evaluate.

$$acc = \frac{\sum_{i=1}^{N} \mathbb{I}(y = \hat{y})}{N} \tag{2.3}$$

Where N is the length of the test set, $\mathbb{I}(\cdot)$ is an indicator function, if $y = \hat{y}$ return 1, otherwise 0.

2.2 Siamese Neural Network

Siamese Neural Network [7] is an architecture commonly used to calculate the similarity between pairs of samples. The network accepts paired samples as input, and then mapped them into a new space by two encoders to obtain the embeddings of the samples. The similarity between the two samples can be measured by distance metrics or a shallow downstream network.

Siamese architecture has been successfully applied in sentence semantic matching task [18]. Typical encoders are Multi-Layer Perceptron (MLP) [19], Recurrent Neural Network (RNN) [15, 20] and Convolutional Neural Network (CNN) [21, 22]. Recent Transformer-based language representation model BERT can also be used as an encoder in Siamese architecture [9]. Distance metrics such as cosine similarity, Manhattan distance, and Euclid distance are often used to measure the relevance between sentence representations. The softmax classifier is to distinguish whether two sentences have a similar meaning. This section will describe two popular sentence semantic matching models that are highly related to this thesis. One is RNN-based Manhattan LSTM (MaLSTM) [15], the other is BERT-based sentence-BERT (SBERT) [9].

2.2.1 Manhattan LSTM Model

The most important component of the Manhattan LSTM Model is the Long Short-term Memory Model (LSTM) [23], which is to alleviate the vanishing gradient problem of

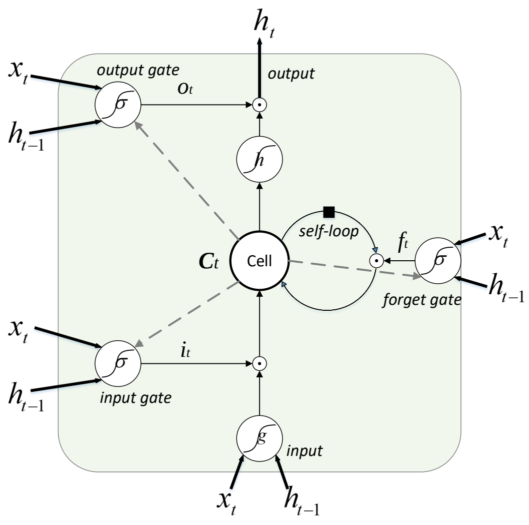


Figure 2.1: The figure shows the structure of LSTM. x_t is the input at the time t and h_{t-1} is the hidden state of the cell at the time $t-1$. C_t is the cell state at the time t the h_t is hidden state at the time t. Image comes from [24].

RNN and capture long-term dependency. Figure 2.1 shows the structure of LSTM. We denote the weights matrices of LSTM as W_i, W_f, W_c, W_o, the bias vectors as b_i, b_f, b_c, b_o, the forward propagation at time step t can be described as following:

$$f_t = \sigma(W_f\ [h_{t-1},\ x_t\ ] + b_f) \tag{2.4}$$

$$i_t = \sigma(W_i\ [h_{t-1},\ x_t\ ] + b_i) \tag{2.5}$$

$$\widetilde{C_t} = \tanh(W_c[h_{t-1},\ x_t\ ] + b_c) \tag{2.6}$$

$$C_t = f_t C_{t-1} + i_t \widetilde{C_t} \tag{2.7}$$

$$o_t = \sigma(W_o\ [h_{t-1},\ x_t\ ] + b_o) \tag{2.8}$$

$$h_t = o_t \tanh(C_t) \tag{2.9}$$

Where C_t is called cell state at time step t, and h_t is called hidden state at time step t. It is believed that the cell state carries long-term memory information and the hidden state represents the working memory.

Figure 2.2 shows the architecture of MaLSTM. Two encoders LSTM_a and LSTM_b share weights and use the last hidden states as the representation of sentences. The similarity between two sentences is measure by a simple similarity function based on Manhattan distance:

$$g(h_t^{(a)}, h_t^{(b)}) = \exp(-\left\|h_t^{(a)} - h_t^{(b)}\right\|_1) \tag{2.10}$$

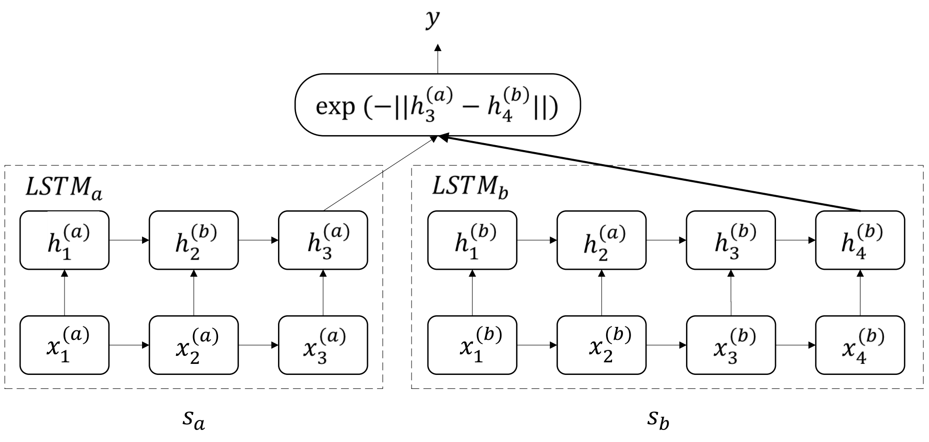


Figure 2.2: The MaLSTM uses LSTMs as its encoders. $\mathrm{LSTM_a}$ and $\mathrm{LSTM_b}$ share weights. The last hidden state is the representation of the input sentence. Mahattan similarity is defined to measure semantic similarity. Image comes from [15].

The reason that the similarity function is based on ℓ_1 norm rather than ℓ_2 norm is because ℓ_2 norm in the similarity function can may to undesirable plateaus in the overall objective function [25].

Mean-squared-error (MSE) is utilized to train the model:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2 \tag{2.11}$$

where $y^{(i)}$ and $\hat{y}^{(i)}$ is the i^{th}-dimension value of y and $\hat{y}$ separately.

Although original MaLSTM is for regression tasks, it can also be used in binary classification tasks [26] by treating 0-1 labels as similarity scores. At inference time, 0.5 is set as the threshold to squeeze the real-value similarity scores into binary labels.

2.2.2 Sentence-BERT

Sentence-BERT is a BERT-based Siamese architecture. BERT [10] as a pre-trained language model has achieved state-of-the-art in many NLP tasks, including the sentence semantic matching task. Two sentences concatenated by a special token are passed to the transformer cross-encoder, and the downstream part of BERT predicts the semantic relevance. However, the procedure is very time-consuming, and a sentence does not have individual representation. SBERT can relieve the problems and provide reasonable individual representations for sentences.

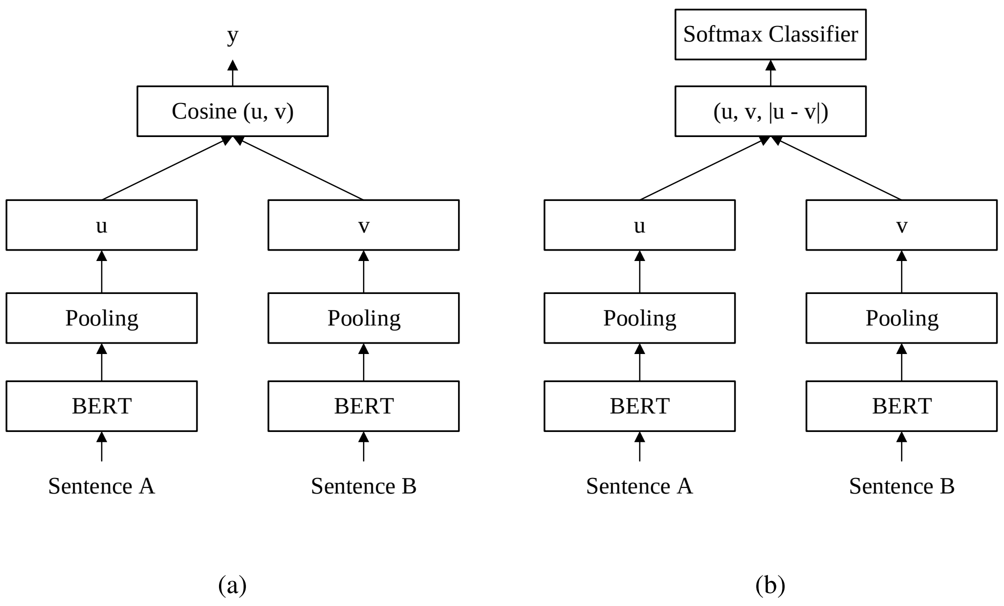


Figure 2.3: Two variants of SBERT. (a) SBERT architecture for regression, predicting the semantic relatedness score. The two BERT encoders share the weights; (b) SBERT architecture for classification. Two BERT encoders have tied weights. Image comes from [9].

Figure 2.3 shows two architectures of SBERT for regression and classification. Both architectures use two BERT with the tied weights to encode sentences. Sentences are encoded as a series of dense vectors through the BERT. The pooling layer pools the output vectors. The pooling strategies can be directly using the output vector corresponding to the `CLS`-token, `MAX`-strategy and `MEAN`-strategy. The sentences are represented as dense vectors after the pooling layer, which is the sentence embeddings. If the task is regression, computes the cosine similarity. If the task is classification, concatenate the sentence representations u, v with the absolute element-wise difference $|u-v|$ and feed to the downstream softmax classifier.

Loss functions for finetuning SBERT are similar to training MaLSTM. If the task is regression, use MSE to train the model. If it is the classification task, use cross-entropy as the loss function. If the dataset is a set of triplet groups (s_a, s_p, s_n), where sentence a is an anchor sentence, sentence p is a positive sentence and sentence n is a negative sentence, use the triplet loss function to tune the SBERT:

$$\mathcal{L}_{triplet} = \max(d(e_a - e_p) - d(e_a - e_n) + \varepsilon, 0) \tag{2.12}$$

Where e_x is the sentence embedding for sentenc x, $d(\cdot)$ is the distance metric and ε is the margin ensures that e_p is at least ε closer to e_a than e_n.

2.3 Binarizing Representation

Sentence representations are usually real-value dense vectors generated from pre-trained deep learning models, reflecting semantic meanings. However, the real-value representation is not memory-efficient, requiring large memory. Consider the situation that in a Q&A system, there are 10 million candidate answers and a 1024 dimensional single-precision vector represents each, then around 40 Gigabit memory will be required to store the vectors [2].

Binarizing the real-value representation is a promising way to reduce the memory footprint [27]. The aim of binarizing sentence representation is to both compress sentence representations and reserve the downstream task performance. Given a set of real-value sentence representations with p-dimension and a downstream task $\mathcal{T}$, the goal is to find a mapping $f : \mathbb{R}^p \mapsto \{0,1\}^m$ such that the performance on $\mathcal{T}$ using the representation after mapping is close to the original performance. Specifically, in the sentence semantic matching task, the distance between binarized representations is expected to be close to that of real-value representations.

2.4 Related Work

Frequently-used methodologies for binarizing representations can be divided into two groups, i.e. data-independent and data-dependent methods [28]. Data-independent methods are usually based on random projections to binarize real-value presentations directly. The representative methods are Locality Sensitive Hashing (LSH) [29] and its variants [30]. This kind of approach defines hash functions to map similar high-dimensional vectors to the same bucket. Data-dependent methods often construct models to learn binarized representation based on data. Semantic hashing [31] is a typical method, which trains an auto-encoder with 30 logistic units for the code layer, and simply thresholds the activities of the 30 code units to get a binary code[3].

Recent research on binarizing sentence representations involves both data-independent approaches and data-dependent approaches. Hard-threshold based approaches [11] is a simple data-independent method to binarize distributed representation. It directly

[2]This case is based on the student's Informatics Project Proposal.

[3]This paragraph is mainly based on the student's Informatics Project Proposal.

binarizes pre-trained representation using a given threshold. To get shorter binary representations, PCA and random projection [32] can be used before binarization. Data-independent approaches are fast but straightforward. However, they are considered less accurate. Shen et al. [11] proposed a data-dependent auto-encoder architecture training with the semantic-preserving regularizer to produce binary embeddings. The semantic-preserving loss explicitly encourages preserving semantic similarity information of the continuous representations [11]. An optimization-based data-dependent approach called Construction and Decomposition algorithm (C&D algorithm) was proposed in [12], which constructs an integer programming to figure out the binary representation.

The above approaches were mainly evaluated from task performance and retrieval speed perspectives. However, they neglected the latency of encoding sentences which is important in real-world sentence matching applications. Besides, all the approaches are based on pre-trained representation rather than training or finetuning the whole model end-to-end. Some research [33, 11] tried to add a binarization layer directly on the InferSent architecture [8] but was reported to perform poorly. However, the experiment results of our architecture show the opposite results.

This project is also related to the binarized neural network (BNN) [13]. BNN binarizes both weights and activation through sign function to compress the model and uses the straight-through estimator (STE) to approximate the gradient of sign function. Although BNN is very small and highly compressive, the accuracy may be influenced. To focus on preserving accuracy, our work only binarizes the activation and uses STE during back-propagation.

Chapter 3

Comparison Framework

Recent work in binarizing sentence representation was evaluated only based on the pre-trained InferSent [8] representations, ignoring the latency of encoding a new-coming sentence. To address the limitations and further evaluate these approaches, this chapter proposes a comparison framework in the sentence semantic matching task.

Figure 3.1 shows the overall framework. The framework is based on two datasets, SICK (Sentences Involving Compositional Knowledge) [16], and MRPC (Microsoft Research Paraphrase Corpus) [17]. Sentence matching models used in this framework are SBERT [9] and MaLSTM [15] introduced in Chapter 2. The grey part of Figure 3.1 indicates the encoders of two Siamese architectures. The encoder outputs are the pre-trained sentence representations prepared for binarization. The downstream part is to calculate the matching score. The evaluation module evaluates different approaches on matching performance, memory footprint and inference latency. The following sections will introduce different modules in detail[1].

3.1 Datasets

The framework chooses the SICK dataset and MRPC dataset mainly for two reasons: *i*) Recent work in binarizing sentence representation for sentence semantic matching uses these two datasets. Thus, it is convenient to make a comparison with others' results; *ii*) These two datasets cover two types of sentence semantic matching tasks since SICK is for the regression task and MRPC is for the classification task.

The SICK [16] dataset consists of 9840 sentence pairs, and each pair is assigned a relatedness score indicating the semantic relevance. The relatedness score is between

[1]This paragraph is based on the student's Informatics Project Proposal.

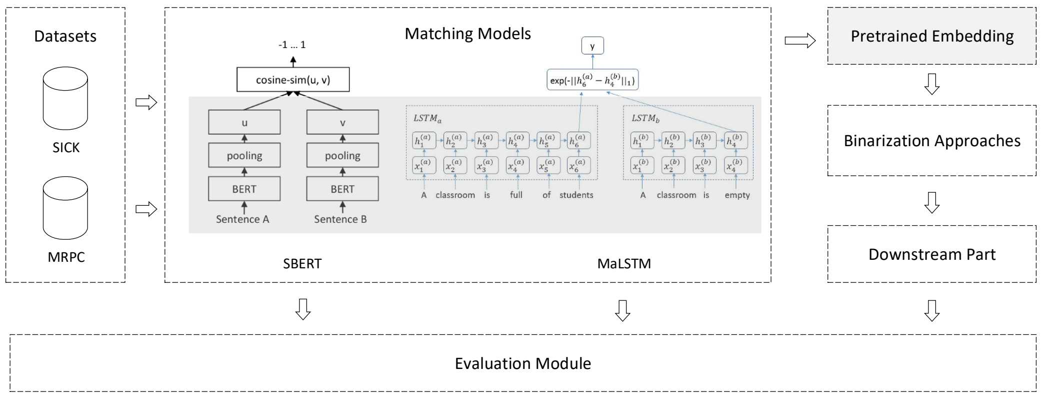


Figure 3.1: The figure shows the overall framework for binarization approaches comparison. The grey part indicates the encoders of two Siamese architectures.

1 to 5, and there are 923 pairs in the [1,2) range, 1373 pairs in the [2,3) range, 3872 pairs in the [3,4) range, and 3672 pairs in the [4,5] range. The dataset is divided into the training set (4439 pairs), validation set (495 pairs) and test set (4906 pairs). The task of SICK is to predict the relatedness score for each sentence pair [16].

The MRPC [17] dataset consists of 4801 sentence pairs and binary label tags for each pair. If two sentences have the same meaning, the label is 1, otherwise 0. The original dataset is divided into a training set (4076 pairs) and a test set (1725 pairs) but without a validation set. We split out a validation set with 816 pairs from the training set for monitoring the training process. Thus there are only 3260 pairs remain in the new training set. The ratio of positive and negative pairs in the three datasets shows in Table 3.1. The data is slightly imbalanced, with more positive pairs than negative pairs.

DATASET	SIZE	NEGATIVE	POSITIVE
Train	3260	32%	68%
Valid	816	34%	66%
Test	1725	34%	66%

Table 3.1: The table shows the ratio of positive and negative pairs in training set, validation set and test set of MRPC dataset.

3.2 Binarization Approaches

We choose the most recent approaches for binarizing sentence representation proposed in [11, 12] to make comparisons, including 3 data-independent approaches and 2 data-dependent approaches.

3.2.1 Hard Threshold-based

Hard Threshold-based binarization approaches binarize continuous sentence representations using a simple sign function. This approach family is data-independent, naively considers the binary representation as an approximation of original representation.

Direct binarization method [11] is a straightforward scheme to binarize sentence representations. It directly binarizes the real-value representations without any transformation. Denote the real-value sentence representation with dimension L as h, binary representation as b and the hard threshold as s. For the i^{th} $(i \leq L)$ dimension in b:

$$b^{(i)} = \mathbf{1}_{h^{(i)} > s} = \frac{\text{sign}(h^{(i)} - s) + 1}{2} \tag{3.1}$$

This approach suffers from information loss badly [34]. Besides, direct binarization with hard threshold can only generate the representation with the same length as the original.

Binarizing after random projection [11] is a method to generate a shorter binary representation. It first does the linear transformation for real-value representation using a random-initialized weight matrix to reduce the dimension, then binarizes the shorter vector through Equation 3.1. The weight matrix $W \in \mathbb{R}^{D \times L}$ is obtained heuristically by sampling from a uniform distribution:

$$W_{i,j} \sim \text{U}(-\frac{1}{\sqrt{D}}, \frac{1}{\sqrt{D}}) \tag{3.2}$$

Where D $(D \leq L)$ is the dimension of the final binary representation. In Shen et al. [11]'s experiment results, this approach is observed to work better than direct binarization.

Binarizing after principal component analysis (PCA) [11] is another method to obtain shorter binary representations. It first reduces the dimensionality of representations, and then the binarizing function is used to binarize each element in the shorter vectors.

Given a set of real-value sentence representations $H = \{h_i\}_{i=1}^{N} \subset \mathbb{R}^L$, centralize the each embeddings as $h_i = h_i - \bar{h}$, where $\bar{h}$ is the mean embeddings. Decompose H as 3

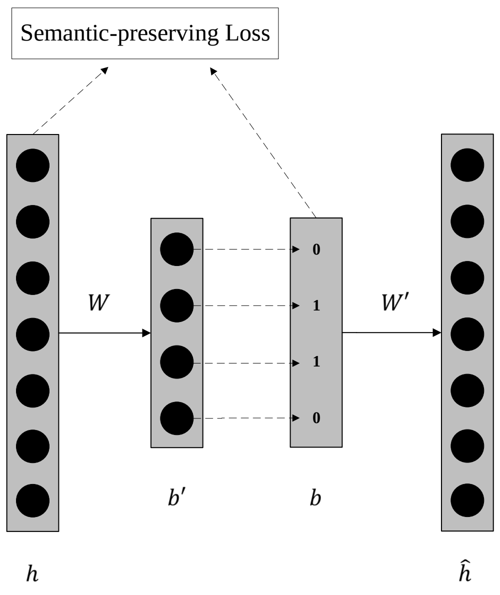


Figure 3.2: The architecture of semantic-preserving autoencoder [11]. h is the continuous representation and W is the weight of the encoder. b is the binarized representation generated from the output of encoder b'. W' is the weight of the decoder and $\hat{h}$ is the reconstructed continuous representation.

matrix:

$$H = U\Sigma V^\top \tag{3.3}$$

where U and V are orthogonal matrices and Σ is the diagonal matrix with L singular values. The correlation matrix is calculated by:

$$HH^\top = U\Sigma V^\top V\Sigma U^\top = U\Sigma^2 U^\top \tag{3.4}$$

The row index list corresponding to the top D value in Σ^2 is denoted as I. Take out the rows in U corresponding to the index list I to construct a projection matrix W. The matrix after PCA is $H' = WH$. Apply the binarize function in each element in H' to get the final binary representation.

3.2.2 Semantic-preserving Autoencoder

Semantic-preserving autoencoder (AE-binary-SP) is a data-dependent approach. Not like the above data-independent methods, this method uses a well-designed loss function to explicitly encourage learning binary representations retaining the semantic information of the real-value representations [11].

The architecture of semantic-preserving autoencoder is shown in Figure 3.2. The real-value representation h is first encoded by the encoder network:

$$b' = \sigma(Wh + k) \tag{3.5}$$

The architecture of the semantic-preserving autoencoder is shown in Figure 3.2. The encoder network first encodes the real-value representation h:

$$b^{(i)} = \mathbf{1}_{b'^{(i)} > s} = \frac{sign(b'^{(i)} - s) + 1}{s} \tag{3.6}$$

Where $b^{(i)}$ is the i^{th} element of b, $b'^{(i)}$ is the i^{th} element of b'. s denotes the binarization threshold and is set 0.5.

Some work has demonstrated that linear decoder is suitable for learning binary representations with encoder-decoder architecture [35, 34, 36], thus a simple linear transformation is used in the decoder to reconstruct the original representations from the binary code, where w' is the weight, and k' is the bias:

$$\hat{h} = W'b + k' \tag{3.7}$$

The objective function for training this model includes two terms, reconstruction loss term and semantic-preserving loss term. The **reconstruction loss** is mean-square error between h and $\hat{h}$:

$$\mathcal{L}_{rec} = \frac{1}{L}\sum_{i=1}^{L}(h^{(i)} - \hat{h}^{(i)})^2 \tag{3.8}$$

Where L is the dimensinality of h. Straight-through estimator [37] is used to estimate the gradients of the sign function.

The **semantic-preserving loss** is inspired by a common knowledge: the relative distance between binary representations should be consistent with the relative distance between real-value representations. For a triple group of sentences $(x_\alpha, x_\beta, x_\gamma)$, the corresponding continuous and binary representations are $(h_\alpha, h_\beta, h_\gamma)$ and $(b_\alpha, b_\beta, b_\gamma)$. When the similarity between h_α and h_β is larger than the it between h_α and h_γ, the Hamming distance between b_α and b_β should be smaller than the it between b_α and b_β. The semantic-preserving loss is defined as:

$$\mathcal{L}_{sp} = \sum_{\alpha,\beta,\gamma} \max(0, l_{\alpha,\beta,\gamma}[d_h(b_\alpha, b_\beta) - d_h(b_\beta, b_\gamma)]) \tag{3.9}$$

The $l_{\alpha,\beta,\gamma}$ is defined as an indicator such that $l_{\alpha,\beta,\gamma} = 1$ if $d_c(h_\alpha, h_\beta) \geq d_c(h_\beta, h_\gamma)$, and $l_{\alpha,\beta,\gamma} = -1$ otherwise. The $d_h(\cdot,\cdot)$ denotes the Hamming distance[2].

[2]This paragraph is taken from the student's Informatics Project Proposal.

Algorithm 1 Decomposition step.

```
Input: $S \in \mathbb{R}^{m \times n}$, $V_p \in \mathbb{R}^{D \times n}$ and $V_p^{-1}$
Output: $B \in \{\pm 1\}^{D \times m}$
1: Calculation: $Q \leftarrow V_p S^\top$;
2: Initialization: $B \leftarrow S V_p^{-1}$;
3: repeat
4:     for one row $b^\top$ in $B$ do
5:         $b \leftarrow sign(q - B'^\top V_p' v)$;
6:     end
7: until B does not change
8: return $B$;
```

The entire loss function combines the two terms with λ_{sp}. $\lambda_{sp} \in [0,1]$ controls the contribution of semantic-preserving loss to the entire loss function:

$$\mathcal{L} = \mathcal{L}_{rec} + \lambda_{sp} \mathcal{L}_{sp} \tag{3.10}$$

3.2.3 Similarity Matrix Construction and Decomposition

Similarity matrix construction and decomposition (C&D algorithm) [12] is another data-dependent approach. It first constructs a similarity matrix between anchor vectors and a batch of inputs, then decomposes the similarity matrix to generate binary outputs.

Anchor vectors are generated heuristically by GlorotNormal [38]. They are like the ruler to measure the nature of the input representations [12]. Thus, the vectors should be consistent in any batch and balanced in spatial distribution, and the number of anchor vectors should be greater than or equal to the dimensionality of vectors [12].

$$V \sim \mathcal{N}(0, \sqrt{\frac{2}{n+L}}) \tag{3.11}$$

Where n is the number of anchor vectors that we want to generate, and L is the dimension of inputs. Note that $V = [v_1, v_2, ..., v_n] \in \mathbb{R}^{L \times n}$, contains n L-dimension anchor vectors. To make sure the anchor vectors are unbiased, we zero-centralise each v_i by subtracting the mean value of each anchor.

To get the D-dimension binary code in the decomposition step, we apply PCA on V and get a smaller matrix, then scale then matrix by $\sqrt{D}$ to get $V_p \in \mathbb{R}^{D \times n}$

Construction step generates the similar matrix between anchor vectors V and a batch of inputs. Denote $H = [h_1, h_2, .., h_m] \in \mathbb{R}^{L \times m}$ as a batch of inputs, whose batch size is m. Each element in the similar matrix S is calculated by:

$$s_{ij} = \frac{h_i^\top v_j}{||h_i||_2 ||v_j||_2} D \tag{3.12}$$

Where s_{ij} is the element in the i^{th} row, the j^{th} column of S. D is the scale factor and also the dimensionality of the final binary representation.

Decomposition step is to solve an optimization problem to get binary representations. Denote the binarized representations as B, formulate an integer programming problem as below:

$$\begin{aligned} &\arg\min_B ||S - B^\top v_p||_F^2, \\ &\text{s.t. } B \in \{\pm 1\}^{D \times m}. \end{aligned} \tag{3.13}$$

Algorithm 1 shows how to solve this programming problem utilizing the discrete coordinate descent method. In the algorithm, $b^\top$ is the l^{th} row of B and B' is the matrix B excluding $b^\top$. Similarly, V_p' is the matrix V_p excluding $v^\top$.

3.3 Downstream Component

For a sentence pair (s_1, s_2), denote the binary representations (b_1, b_2), the length of vector D. If the task is predicting semantic relatedness (corresponding to the SICK dataset), we define a simple Hamming similarity as the downstream component:

$$\text{sim}_h(b_1, b_2) = 1 - \frac{d_h(b_1, b_2)}{D} = \frac{D - \sum_{i=1}^{D} (b_1^{(i)} \oplus b_2^{(i)})}{D} \tag{3.14}$$

Where $b_1^{(i)}$ and $b_2^{(i)}$ are the i^{th} element of $b_1^{(i)}$ and $b_2^{(i)}$ respectively, $d_h(\cdot, \cdot)$ denote the Hamming distance.

If the task is classification, which is to determine whether two sentences express the same meaning (corresponding to MRPC dataset), we define a softmax classifier as the downstream component:

$$f(b_1, b_2) = \text{softmax}(W[b_1, b_2, |b_1 - b_2|] + \beta) \tag{3.15}$$

Where W is the weight and β is the bias. This softmax classifier is also used in SBERT [9] architecture and InferSent [8] architecture and is observed suitable for sentence semantic matching.

3.4 Evaluation Module

The evaluation module assesses representations from memory footprint, latency and matching performance perspectives. The test set of the framework is consist of sentence pairs. For each sentence pair, we treat one sentence as a *query*, and the other one as the *value*[3]. For a test set with length N, we can get N queries and N values. The values will be pre-computed as a set of vectors stored in the main memory.

Memory Footprint refers to the memory footprint of pre-computed vectors in the main memory. It is an important criterion to judge whether the representations are memory-efficient. In the main memory, we define that the continuous vector is stored in 32-bits float type, and the binarized vector is stored in 8-bits boolean type. We record the actual main memory footprint of pre-computed representations as our concerned footprint.

Matching Latency is the average time needed for the model calculating a matching score for one sentence pair. In a test set, the time needed to process all sentence pairs is defined as:

$$t_{all} = t_{repr} + t_{match} \tag{3.16}$$

Where t_{repr} is the time needed to compute the representation of the queries, t_{match} is the time needed to compute the matching scores between queries and pre-computed values. We define the matching latency t as:

$$t = \frac{t_{all}}{N} \tag{3.17}$$

Where N refers to the number of sentence pairs.

Matching Performance[4] is evaluated differently on different datasets. The SICK dataset uses a 5-point rating scale to measure the sentence relatedness score. Thus, we use Pearson correlation and Spearman correlation to measure the correlation between the predicted and true scores. MRPC dataset uses 0 and 1 to tag matching or not. The accuracy is used to measure the matching performance.

3.5 Summary

In this chapter, we build up the overall framework to compare different binarized representations. The chosen two datasets cover two types of sentence semantic matching

[3]Concepts of query and value come from information retrieval.

[4]This part is based on the student's Informatics Project Proposal.

tasks. We use pre-trained SBERT and random-initialized MaLSTM to produce continuous sentence representations, enriching the testbed for binarized approaches. The evaluation method for matching latency considers the time that model encodes sentences, making it better to simulate real-world applications.

Chapter 4

An End-to-end Siamese Network for Learning Binary Representations

4.1 Motivation

In Chapter 3, we introduced several approaches to binarize sentence representations. However, all the methods follow the same pattern, binarizing the pre-trained sentence representations rather than generating binary representations end-to-end. A potential problem of this pattern is, the quality of binary representation seriously depends on the nature of the original continuous representations, such that we have to try many approaches to find a suitable one obtaining good binarized representations. This is costly and not clever enough.

An intuitive scheme to address this issue is to train a model combining with the downstream task in an end-to-end manner to generate binary representations. This idea is relatively common in the Computer Vision field (CV) [39, 40, 41], and is also claimed to be effective in word embedding binarization [14]. Thus, we hypothesize that end-to-end training can provide guidance to generate high-quality binary representations.

The semantic-preserving autoencoder [11] described in Chapter 3 inspired us, too. The similar structure of AE-binary-SP is easy to be appended at the top of a Siamese network, and the semantic-preserving loss is also quite suitable for training the Siamese architecture. Based on this, we designed an architecture called **BinSiamese**, which is an end-to-end Siamese network for learning binary representations.

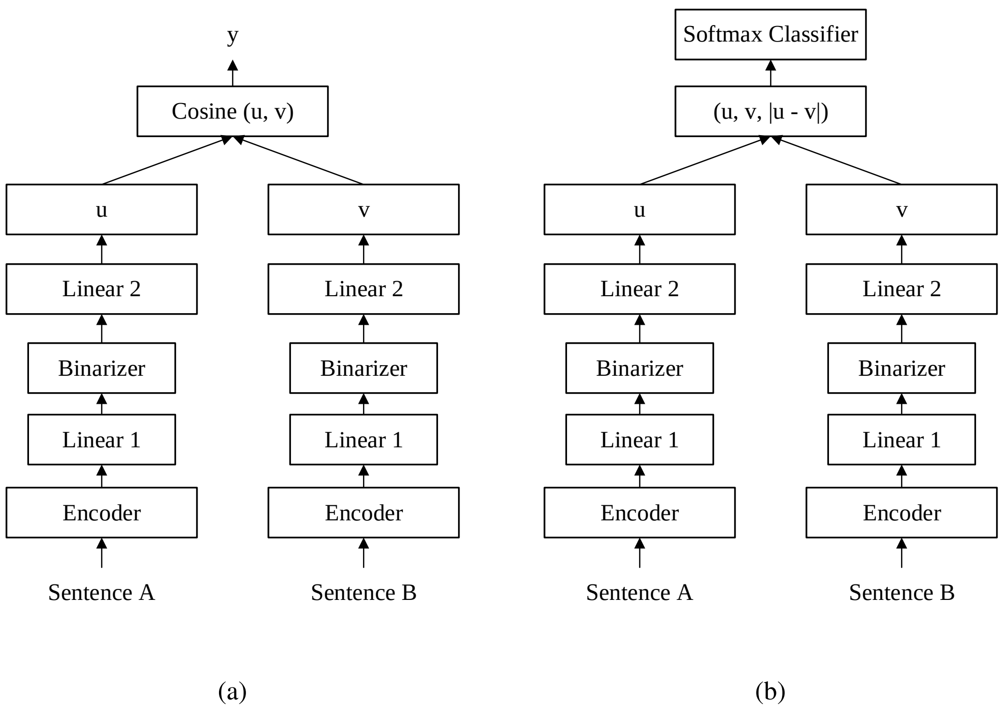


Figure 4.1: (a) BinSiamese architecture for predicting semantic relatedness. (b) BinSiamese architecture for classification. In each architecture, two branches have tied weights.

4.2 Model Design

4.2.1 Overall Architecture

Figure 4.1 shows two BinSiamese architectures, variant (a) is for the regression task and variant (b) is for the classification task. Each branch of the network consists of an encoder, two linear layers and a binarizer. The encoder is typically pre-trained LSTM or BERT. All weights in two branches are tied. Similar to SBERT, the cosine similarity is used to measure the semantic relatedness between two sentences, and the softmax classifier determines whether two sentences express the same meaning.

For a sentence pair (s_1, s_2), the encoder layer generates the L-dimension embeddings h_1, h_2 for s_1, s_2 respectively, then feeds them into the first linear layer:

$$\begin{aligned} o_1 &= \tanh(\mathrm{BN}(W_1 h_1 + k_1)) \\ o_2 &= \tanh(\mathrm{BN}(W_1 h_2 + k_1)) \end{aligned} \tag{4.1}$$

Where $\mathrm{BN}(\cdot)$ is batch normalization [42], W_1 is the weight matrix and k_1 is the bias. Note that W_1 is a $D \times L$ matrix, where D is the dimension of our desired binary representation and $D \leq L$.

The binarizer applies binarize function on each element in o_1 and o_2 to get binary representations:

$$\begin{aligned} b_1 &= \text{bin}(o_1) = [\mathbf{1}_{o_1^{(1)}>0}, \mathbf{1}_{o_1^{(2)}>0}, \ldots, \mathbf{1}_{o_1^{(1)}>0}]^\top \\ b_2 &= \text{bin}(o_2) = [\mathbf{1}_{o_2^{(1)}>0}, \mathbf{1}_{o_2^{(2)}>0}, \ldots, \mathbf{1}_{o_2^{(1)}>0}]^\top \end{aligned} \tag{4.2}$$

In the back-propagation, because the gradient of $\text{bin}(\cdot)$ vanishes almost everywhere, we use straight-through estimator (STE) [37] to directly pass the gradients of o_1 and o_2 through the hard tanh function:

$$\text{Htanh}(x) = \text{Clip}(x, -1, 1) = \max(-1, \min(1, x)) \tag{4.3}$$

This is also used in bianrized neural network [13] and AE-binary-SP [11]. The binary codes b_1 and b_2 is our desired binary representations.

The binary codes are fed into the second linear layer to obtain the continuous representations u and v.

$$\begin{aligned} u &= \tanh(\text{BN}(W_2 b_1 + k_2)) \\ v &= \tanh(\text{BN}(W_2 b_2 + k_2)) \end{aligned} \tag{4.4}$$

Where W_2 is the $L \times D$ weights and k_2 is the bias. b_1 and b_2 are binary vector and the u and v can be seen as the combination of rows in W_2 after non-linear transformation, so the W_2 is similar to the codebook in some vector quatization methods [43].

After getting the continuous representations, cosine similarity or softmax classifier handles regression task or classification task respectively:

$$\hat{y} = \text{cosine}(u, v) = \frac{u^\top v}{|u| \times |v|} \tag{4.5}$$

$$\hat{y} = \text{softmax}(W_3[u, v, |u - v|] + \beta) \tag{4.6}$$

Where w_3 is the weight matrix of the classifier and β is the bias.

At inference time, we take off the part above the binarizer and only remain the encoder layer, the first linear layer and the binarizer. The output of the remaining structure is our desired binary representations b_1 and b_2.

4.2.2 Encoder Layer

The BinSiamese architecture is flexible to use different encoders. The encoder layer is typically a pre-trained model that embeds sentences into continuous space. Inspired by the design of MaLSTM and SBERT, we figure out two encoder layers fitting into the BinSiamese architecture, one is the LSTM encoder, the other is the BERT encoder.

The LSTM encoder is the LSTM layer taken from the pre-trained MaLSTM. The last hidden states of the LSTM is treated as sentence embeddings. We call the BinSiamese applying LSTM encoder the **BinLSTM**. Our initial attempts demonstrate that the LSTM encoder layer can only be applied to variant (a). When applying it to variant (b), we observed a massive drop in matching performance. Although variant (b) is not available, variant (a) can also deal with binary classification tasks by treating the 0-1 label as the similarity score, similar to MaLSTM.

The BERT encoder is the pre-trained BERT with a pooling layer on the top. This is the same as the structure in SBERT. The average strategy is chosen in the pooling layer. BERT encoder can be used in both variant (a) and variant (b). We call the BinSiamese applying BERT encoder the **BinSBERT**.

4.3 Loss Function

Two objectives should be guaranteed to train BinSiamese. The first objective is, the continuous representations need to reflect the semantic meaning of corresponding sentences. The second objective is that the binary representations also need to reflect the semantic meaning of corresponding sentences. The two objectives are related because continuous representations are generated from binary representations, and binary representations are also updated utilizing the information backpropagated from the continuous representations during training.

To guarantee the first objective, we utilize the cosine similarity $\hat{y}$ (or the predicted label) of continuous representations and the ground-truth similarity y (or label) to construct loss functions. If the task is to predict semantic similarity, then the loss function is MSE:

$$\mathcal{L}_1 = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2 \tag{4.7}$$

If the task is to predict the class label, then the loss function is cross-entropy:

$$\mathcal{L}_1 = -\sum_{i=1}^{N} y^{(i)}\log(\hat{y}^{(i)}) \tag{4.8}$$

where $y^{(i)}$ and $\hat{y}^{(i)}$ is the i^{th}-dimension value of y and $\hat{y}$ separately. Note that when training BinLSTM to deal with binary classification tasks, loss function $\mathcal{L}_1$ should also be MSE.

For the second objective, we apply the semantic-preserving loss [11] introduced in Chapter 3. This loss function encourages the binary representations to follow the semantic meaning of the continuous representations. Given a triple group of sentences$(x_\alpha, x_\beta, x_\gamma)$ whose continuous and binary representations are $(u_\alpha, u_\beta, u_\gamma)$ and $(b_\alpha, b_\beta, b_\gamma)$ respectively. We modify the Equation 3.9 as:

$$\mathcal{L}_2 = \sum_{\alpha,\beta,\gamma} \mathrm{Relu}(\mathrm{sign}(d_c(u_\alpha, u_\beta) - d_c(u_\beta, u_\gamma)) \times (d_h(b_\beta, b_\gamma) - d_h(b_\alpha, b_\beta)))^2 \tag{4.9}$$

where $d_c(\cdot,\cdot)$ is the cosine similarity, $d_h(\cdot,\cdot)$ is the Hamming distance, and $\mathrm{sign}(\cdot)$ is the signum function. $\mathrm{Relu}(\cdot)$ refers to Rectified Linear Unit, which is to prune the negative part of the sign function.

The entire loss function combines $\mathcal{L}_1$ and $\mathcal{L}_2$ with the weight λ_{sp}:

$$\mathcal{L} = \mathcal{L}_1 + \lambda_{sp}\mathcal{L}_2 \tag{4.10}$$

where λ_{sp} controls how much the semantic-preserving loss contributes to the entire loss and $\lambda_{sp} \in [0,1]$.

Chapter 5

Experiments & Results

5.1 Experiment Setup

All experiments are based on the DICE computer in the School of Informatics. The GPU we used is GTX 1080 Ti, and the CPU is Intel Core i5-8500. Python 3.8.1 and torch 1.7.1 are used for model implementation. We put all matrix operations on GPU to accelerate training and testing. All models are evaluated in this environment.

5.1.1 Real-value Representation

The real-value representations are generated from pre-trained MaLSTM and SBERT and are the ingredients of the comparison experiments. Thus we need to ensure the quality of the representations.

For MaLSTM, we proposed a hyper-parameter searching experiment to tune the dropout rate (p) between the embedding layer and the LSTM layer the learning rate (lr). We fix the dimension of representation as 1024, set $p \in [0.1, 0.2, 0.3, 0.4, 0.5]$ and $lr \in [0.01, 0.001, 0.001]$, use grid search to figure out the best settings. We monitor the loss on the validation set and choose the setting corresponding to the lowest loss. The best settings we find in SICK dataset and MRPC dataset are: $lr = 0.001$, $p = 0.3$ and $lr = 0.001$, $p = 0.1$ respectively. We then use the best setting to train the MaLSTM. The optimizer used in training is Adam [44]. Additionally, We use the `ReduceLROnPlateau`[1] scheduler to control the learning rate. The reduction factor is set to 0.1 and patience is set to 5. The Early stopping mechanism is also applied, with patience 20. The max training epoch is set to 100.

[1] https://pytorch.org/docs/master/optim.html#torch.optim.lr_scheduler.ReduceLROnPlateau

DATASET	SICK-*dim*	MRPC-*dim*
MALSTM	$\{1024, 512, 256, 128, 64\}$	$\{1024, 512, 256, 128\}$
SBERT	$\{768, 384, 192, 96, 48, 24\}$	$\{768, 384, 192, 96, 48\}$

Table 5.1: The dimension numbers for binarized representations. In SICK and MRPC: when binarizing MaLSTM representation, the dimension number is set as $\{1024, 512, 256, 128, 64\}$ and $\{1024, 512, 256, 128\}$ respectively; when binarizing SBERT representations, the dimension number is set as $\{768, 384, 192, 96, 48, 24\}$ and $\{768, 384, 192, 96, 48\}$ respectively.

We load the `distilbert-base-uncased`[2] model in BERT layer for the regression task on the SICK dataset and load `nli-roberta-base-v2`[3] in the BERT layer for the classification task on the MRPC dataset. We finetune the models on our datasets for 10 epochs, using the default settings of `sentence-transformers`[4] library. We monitor the Pearson correlation or accuracy on the validation set during training and save the best-performance model.

After training the models, we take out their encoders to generate continuous sentence representations for two datasets. The dimension of representations from MaLSTM is 1024, and from SBERT is 768.

5.1.2 Details of Binarization

We set different dimension numbers for binarized representations (excluding the direct binarization approach) to make comparisons. The configuration is shown in Table 5.1.

The binarization threshold of hard threshold-based approaches is set to 0 since values in continuous representations are all between -1 and +1 and zero-centralized. To train semantic-preserving autoencoder, we use the Adam optimizer with 1×10^{-5} learning rate. The hyper-parameter λ_{sp} is selected from $\{0, 0.1, 0.2, 0.5, 0.8, 1\}$ by monitoring the Pearson correlation or accuracy of the downstream task on the validation set. No additional setting is needed for the C&D algorithm.

Our proposed two models BinLSTM and BinSBERT, are trained using transfer learning strategy. For BinLSTM in both two datasets, we load the LSTM weights from the pre-trained MaLSTM and use Adam optimizer with a small learning rate

[2]This is a distilled BERT with a smaller size.
[3]This is a RoBERTa model with an average pooling layer, pre-trained on the NLI dataset.
[4]https://www.sbert.net/index.html

1×10^{-5} to finetune. The scheduler `ReduceLROnPlateau` is also used to schedule the learning rate, with the reduction factor of 0.1 and patience 5. Early stopping is used to terminate training, with patience 20 and max epoch number 300. For BinSBERT, we load `distilbert-base-uncased` for regression and `nli-roberta-base-v2` for classification. For regression, we use a small learning rate 2×10^{-5} to finetune. For classification, we freeze the encoder layer and use the learning rate 1×10^{-4} to train 20 epochs, and then unfreeze the encoder layer and finetune with the learning rate 1×10^{-5} for 20 epochs. The other hyperparameters are the same as the default configuration on SBERT.

To determine the best λ_{sp} for our models under different dimensionality of binary representations, We choose the best λ_{sp}s setting corresponding to the best performance of binary representation on the validation dataset.

5.1.3 Evaluation

We evaluate the performance of different binarization methods from memory footprint, matching latency and matching performance following the comparison framework introduced in Chapter 3. We experiment 10 times and record the average matching latency and its standard deviation. For the binarizing after the random projection method and C&D algorithm, we run experiments with 10 different random seeds and record the average matching performance and its standard deviation to avoid the random affection.

5.1.4 Baselines

We consider the MaLSTM and SBERT as baselines to compare with the binarization methods chosen in the comparison framework. For our proposed models BinLSTM and BinSBERT, we treat both continuous embedding methods and binarization methods as baselines to make comparisons. All methods can be compared with each other fairly among our comparison framework.

5.2 Performance Comparisons

The evaluation results on SICK and MRPC are shown in Table 5.2 and Table 5.3 respectively. Due to the page limitation, we only report the best result of different methods. The experiment results under different dimension numbers are in Appendix A.

MODEL	DIM	PEARSON	SPEARMAN	LATENCY	MEM
MaLSTM	1024	.824	.757	$103.42 \pm .60$	19.16
HT-binary	1024	.595	.499	$103.90 \pm .15$	4.79
Rand-binary	1024	$.726 \pm .006$	$.649 \pm .012$	$104.69 \pm .54$	4.79
PCA-binary	128	.674	.669	$106.30 \pm .20$	0.60
C&D	512	$.380 \pm .028$	$.336 \pm .016$	10912.22 ± 1109.67	2.40
AE-binary-SP	1024	.759	.700	$105.63 \pm .76$	4.79
BinLSTM	256	**.785**	**.714**	$102.85 \pm .03$	1.20
SBERT	768	.881	.834	625.78 ± 2.89	14.37
HT-binary	768	.861	**.828**	611.75 ± 5.35	3.59
Rand-binary	768	$.858 \pm .001$	$.824 \pm .002$	614.37 ± 5.49	3.59
PCA-binary	48	.722	.750	611.43 ± 4.25	0.23
C&D	768	$.860 \pm .001$	$.827 \pm .001$	27726.70 ± 1351.04	3.59
AE-binary-SP	768	.857	.813	617.45 ± 6.03	3.59
BinSBERT	768	**.866**	.815	618.21 ± 2.18	3.59

Table 5.2: The table shows the performance of the continuous representations and binary representations on the SICK dataset. The unit of latency is μs and the unit of memory footprint (MEM) is MB. MaLSTM and SBERT mean the real-value representations generated from MaLSTM and SBERT respectively. The rows following these two rows are the binarization approaches based on their representations or the pre-trained encoder. HT-binary refers to direct binarization. Rand-binary means binarizing after random projection. PCA-binary means binarizing after PCA. C&D refers to the C&D algorithm. The λ_{sp} of AE-binary-SP is 0.2 and 0.8 under MaLSTM and SBERT respectively. The λ_{sp} of BinLSTM and BinSBERT is 0.1 and 1.0 respectively.

On both the SICK and MRPC datasets, our proposed models are very competitive. The binary representations generated from BinLSTM achieve the best matching results with only 256 dimensions, compared to the other binary representations based on the continuous representations from MaLSTM. The latency of using BinLSTM's representations to match is also competitive. Besides, comparing to MaLSTM's continuous representations, BinLSTM's binary representations achieve the same accuracy on MRPC and preserve above 95% correlation on SICK while reducing its 16 times memory footprint. BinSBERT's binary representations get the highest Pearson correlation and accuracy on SICK and MRPC datasets compared to other binarized representations. On MRPC, it even outperforms SBERT. Although the dimensionality is the same as the continuous representations', the memory footprint is 4 times less than the continuous. The matching latency when using BinSBERT is higher than most binarization methods but is still within the acceptable range.

The C&D algorithm is not suitable to be applied in real-world applications due to its extremely high latency[5], although its matching performance is not bad on MRPC. It takes more than 6 times as long as other methods to encode sentences and handle the downstream task. Most of the time consumes on generating binarized embeddings because of an optimization loop in the algorithm. The AE-binary-SP and hard threshold-based approaches are still very practical binarization methods. The binary representations generated by the semantic-preserving autoencoder perform pretty well on both datasets, especially when using MaLSTM's continuous representations. This demonstrates the power of semantic-preserving loss function. On the MRPC dataset, the direct binarization approach achieves the best Spearman correlation based on SBERT' representations, although it is very simple. The binarizing after random projection/PCA method also performs relatively well (PCA is slightly inferior).

An interesting finding is, binarized SBERT's representations can outperform the original continuous representations on MRPC, even using the simple direct binarization method. This is also observed in [11, 45]. A possible explanation of the improvement generalization performance is that binarization can reduce the learned noise to prevent overfitting [46], which is similar to the model pruning. We also observe that in different downstream matching tasks and under different pre-trained representations, the generalization performance of non-end-to-end binarization approaches can be various. For example, on SICK, HT-binary's performs very close to SBERT when using the SBERT's representations. In contrast, on the MRPC dataset under MaLSTM's representations, the downstream softmax classifier predicts all labels as 1 when using HT-binary's representations. However, our end-to-end models do not suffer from this issue and perform relatively stable, showing end-to-end learning approaches' adaptivity.

[5]This observation is not consistent with the original paper. We provide more details about the implementations and some potential explanations in Appendix B.

MODEL	DIM	ACC	LATENCY	MEM
MaLSTM	1024	.711	$102.39 \pm .11$	6.74
HT-binary	1024	.665	$102.97 \pm .06$	1.69
Rand-binary	1024	$.690 \pm .008$	105.59 ± 4.84	1.69
PCA-binary	512	.692	$106.14 \pm .14$	0.84
C&D	1024	$.681 \pm .008$	64823.70 ± 10002.39	1.69
AE-binary-SP	1024	.700	$107.60 \pm .23$	1.69
BinLSTM-SP	256	**.711**	105.09 ± 3.94	0.42
SBERT	768	.743	2469.40 ± 5.99	5.05
HT-binary	768	.759	2464.35 ± 5.74	1.26
Rand-binary	768	$.760 \pm .008$	2506.06 ± 6.25	1.26
PCA-binary	384	.760	2465.00 ± 6.00	0.63
C&D	768	$.757 \pm .005$	30742.93 ± 2885.93	1.26
AE-binary-SP	768	.766	2494.80 ± 5.58	1.26
BinSBERT	768	**.773**	2573.78 ± 2.68	1.26

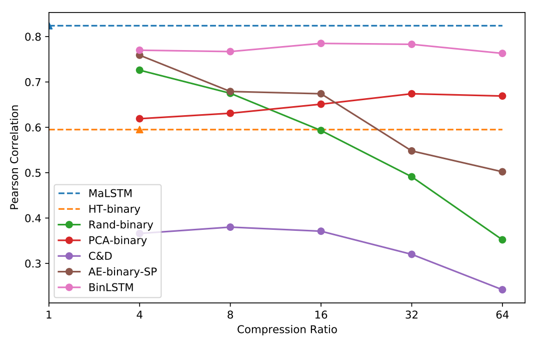
Table 5.3: The table shows the performance of both continuous and binary representations on the MRPC dataset. The notations are the same to the Table 5.2. The λ_{sp} of AE-binary-SP is 0.8 and 1.0 under MaLSTM and SBERT respectively. The λ_{sp} of BinLSTM and BinSBERT is 0.8 and 1.0 respectively.

5.3 Ablation Study

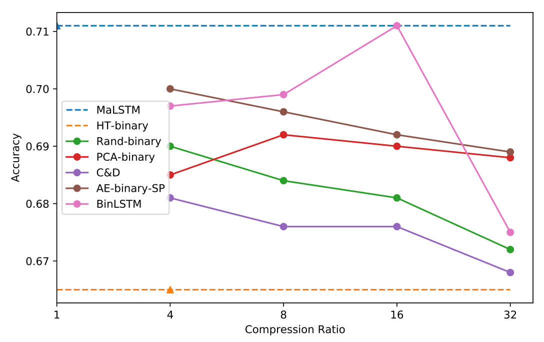
5.3.1 Effect of Representation Dimension

To investigate the sensitivity of binary representation to the dimension, we plot the test matching performance of different methods on SICK and MRPC datasets under different dimensionalities. The compression ratio refers to the ratio of continuous representations' the memory footprint and the representations' after compression, reflecting the dimensional change. Note that in Figure 5.1, we use dotted lines to indicate the performance of continuous representations and HT-binary as the reference, but their dimension is fixed.

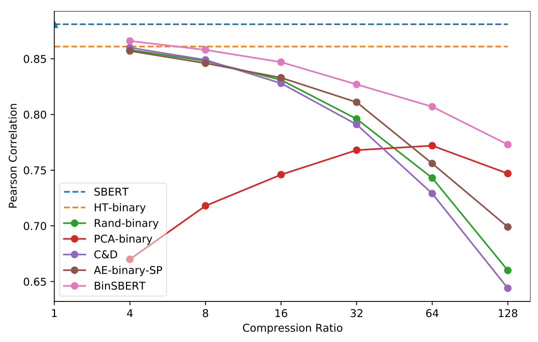
For BinLSTM, with the dimension keeps reducing, its matching performance first improves and then falls. BinLSTM is less sensitive to dimension on the SICK datasets than on the MRPC dataset and can perform very close to the continuous representations even under a very high compression ratio. For BinSBERT, on both the regression



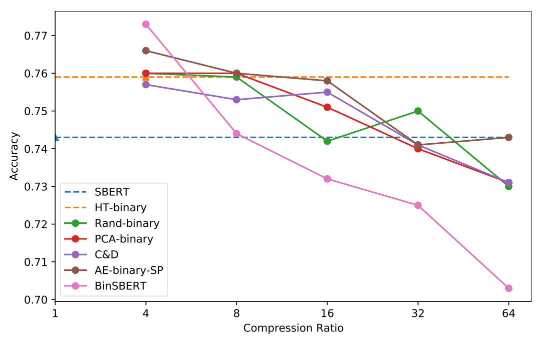
(a) MaLSTM-SICK



(b) MaLSTM-MRPC



(c) SBERT-SICK



(d) SBERT-MRPC

Figure 5.1: The figure shows the test matching performance of different approaches under different compression ratios. The compression ratio refers to the ratio of the memory footprint of continuous representations and the representations' after compression, which also reflects the dimensional change. The dotted lines indicate the continuous representations or HT-binary whose dimensionality couldn't change. Sub-figure (a) and (b) reflect the performance of BinLSTM and methods using MaLSTM's representations on SICK and MRPC. Sub-figure (c) and (d) reflect the performance of BinSBERT and methods using SBERT's representations on SICK and MRPC.

and the classification task, its matching performance decreases with reducing dimensionality. BinSBERT is more sensitive to the classification task. Its performance drops dramatically and quickly becomes worse than all the others in the MRPC dataset.

For the PCA-binary, the matching performance shows a trend of first increasing and then decreasing as the compression ratio increases, while the C&D algorithm shows the same trend only on the SICK dataset when using MaLSTM's representations. For Rand-binary and AE-binary-SP, their sensitivities are similar, and longer binary representations can always achieve better results. Rand-binary, AE-binary and our proposed model perform relatively close when the binary representations have the same dimension with the continuous. We also observe that HT-binary is a solid baseline to produce the same-dimension binary representations when using SBERT's continuous representations.

Based on the observations, we summarize the suggestions for choosing binarization approaches for the sentence semantic matching task. If the matching accuracy is extremely desired, high-dimension binary representations from BinSBERT are the first choice. HT-binary, AE-binary-SP and Rand-binary are also good choices if pre-trained SBERT representations are available. If the low latency and small memory footprint are the priority while accuracy is the second, BinLSTM would be better. PCA-binary could also be a good candidate if MaLSTM's representations are available.

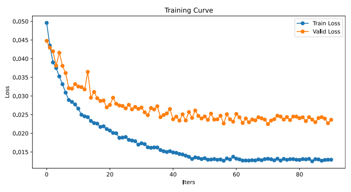
5.3.2 Effect of Semantic-preserving Loss

λ_{sp}	0.0	0.1	0.2	0.5	0.8	1.0
PEARSON	0.764	0.770	0.761	0.753	0.760	0.751
ACCURACY	0.679	0.689	0.710	0.697	0.705	0.689

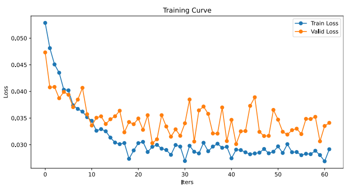
Table 5.4: The table shows the matching performance of BinLSTM under different values of λ_{sp} on both the SICK and MRPC datasets. Pearson correlation and accuracy are corresponding to the matching performance on SICK and MRPC respectively.

To explore the influence of the coefficient of semantic-preserving loss λ_{sp}, we test the matching performance of BinLSTM under different values of λ_{sp} on both the SICK and MRPC datasets as shown in Table 5.4. The dimension of binary representations is 1024. The matching performance shows a trend of first increasing and then decreasing as the λ_{sp} increases. The high λ_{sp} may cause a drop in performance and result worse than $\lambda_{sp} = 0$ according to our results.

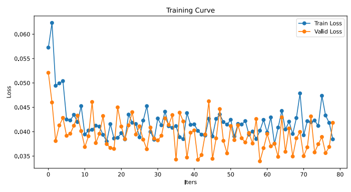
The Loss function is discontinuous and non-convex if the $\lambda_{sp} \neq 0$, hence it is challenging to be optimized [47]. We plot the training curves of BinLSTM using different λ_{sp}s on the SICK dataset as shown in Figure 5.2. We can see both training loss and validation loss fluctuate as the λ_{sp} increases. This phenomenon is not unique for Bin-Siamse architecture as we also observe the same in training AE-binary-SP as shown in Figure 5.3. We believe unstable gradients of semantic-preserving loss cause the fluctuation. Thus, it is necessary to save the model corresponding to the best performance on the validation set during training epochs, and λ_{sp} should be chosen carefully.



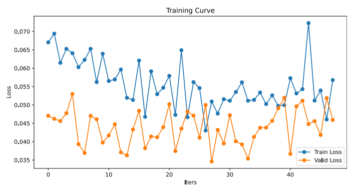
(a) $\lambda_{sp} = 0.0$



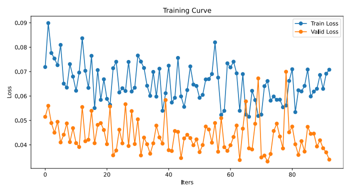
(b) $\lambda_{sp} = 0.1$



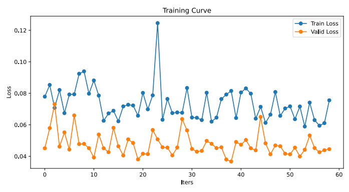
(c) $\lambda_{sp} = 0.2$



(d) $\lambda_{sp} = 0.5$

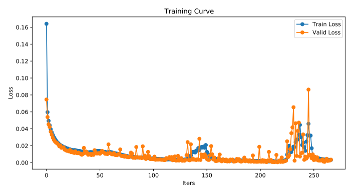


(e) $\lambda_{sp} = 0.8$

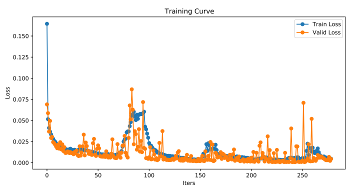


(f) $\lambda_{sp} = 1.0$

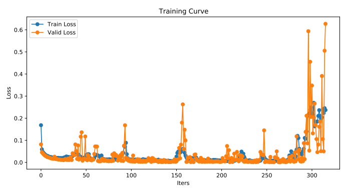
Figure 5.2: The figure shows the training curve of BinLSTM using different λ_{sp}. The dataset that the model trained on is SICK. The dimension of binary representations is 1024.



(a) $\lambda_{sp} = 0.2$



(b) $\lambda_{sp} = 0.5$



(c) $\lambda_{sp} = 0.8$

Figure 5.3: The figure shows the training curve of AE-binary-SP using different λ_{sp}. The dataset that the model trained on is SICK. The dimension of binary representations is 1024.

5.4 Case Study: Nearest Neighbour Retrieval

An important application of sentence semantic matching is information retrieval. K-nearest neighbour is one of the simplest algorithms to retrieve by calculating the distance between query and keys to find the top-k close values. We conduct the nearest neighbour retrieval case study based on the SICK dataset to intuitively assess the quality of binary representations produced by our proposed models. We take out one column of sentences in SICK as queries and leave the other column as answers. Duplicates and sentences that are the same as the query are removed from the answers.

Table 5.5 and Table 5.6 shows the some retrieval cases of BinLSTM vs MaLSTM and BinSBERT vs SBERT separately. We present the three most similar sentences of each query. Most of the selected sentences based on binary representations are the same as those based on the continuous. The semantic meaning of the selected sentences is also very similar to the query sentences. Some cases demonstrate that our proposed models can learn the semantic meaning of sentences rather than memorising sentences rotely. For example, in the last case of Table 5.5, the selected sentence use ”gathering around” to express the meaning of ”sitting around”; in the last case of Table 5.6, the selected sentence contains ”seabird”, which is a similar animal to ”duck”.

5.5 Discussion

Our experiment results demonstrate that learning binary representation end-to-end improves the downstream performance in the sentence semantic matching task. However, Shen et al. [11] claim directly training the binary representations end-to-end leads to inferior results. We observe the same drop in our initial attempt. We attribute it to the architecture and the training strategy.

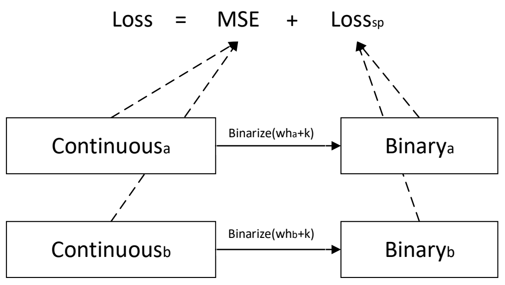


Figure 5.4: Our initial attempt append a binarization layer over the structure of MaLSTM or SBERT and train the model combining semantic-preserving loss.

Our initial attempt appended a binarization layer over the structure of MaLSTM or SBERT and trained the model combining semantic-preserving loss, as shown in Figure 5.4. But we observed the semantic-preserving loss shrinks to 0 quickly during training, and all binary representations are the same. We believe this model structure could not build the direct connection between binary and continuous representation. Thus the binary representation is excluded from the calculation of MSE loss, such that the semantic-preserving loss becomes an independent term in the entire loss function. The semantic-preserving loss will be 0 when binary representations are the same. Thus the model will be optimized to the state that only produces the same binary representations. Our model addresses this problem by utilizing binary representation to produce continuous representations.

In addition, we observe a significant drop if our proposed model does not use the pre-trained layer. Therefore, we argue that the pre-trained encoder can provide a good starting point to optimize, cmake the model get trained better.

Hamming Distance (binary)	Mahattan Similarity (continuous)
Query: A cat is playing a keyboard	
The cat is playing keyboards	The cat is playing keyboards
A cat is playing keyboards	A cat is playing keyboards
A cat is playing a piano	A cat is playing a piano
Query: Two dogs are wrestling and hugging	
A dark black dog and a light brown dog are playing in the backyard	A dog which is brown and a black one are running in the grass
Two dogs are playing in a forest	Two dogs are fighting for a frisbee in a lake
Some dogs are playing in a stream	A dog with long hair and a red vest is sitting in the grass
Query: A girl is cutting butter into two pieces	
A girl is cutting the butter into two pieces	A girl is cutting the butter into two pieces
Some ingredients are being separated in a bowl by a person	Some ingredients are being separated in a bowl by a person
Some ingredients are being mixed in a bowl by a person	A girl is melting two pieces of butter together
Query: A dog is leaping high in the air and another is watching	
The dog is jumping for a frisbee in the snow	A dog is jumping for a frisbee in the snow
A dog is jumping for a frisbee in the snow	The dog is jumping for a frisbee in the snow
A dog is jumping into the air in the country	A dog is jumping in the air
Query: Several people are sitting around a fire at night	
Several people are gathered around a fire at night	People are sitting around a bonfire at night
People are clustered around a bonfire at night	People are clustered around a bonfire at night
People are sitting around a bonfire at night	A few people are floating on a raft

Table 5.5: The retrieval cases using representations produced by BinLSTM and MaLSTM.

Hamming Distance (binary)	Cosine Similarity (continuous)
Query: A group is not riding the current in a raft	
People are not riding and paddling a raft	People are not riding and paddling a raft
The man is not going into the water	The orange rescue boat is not rushing across the water
The person is not going into the water	A group of friends are riding the current in a raft
Query: A person is wearing a hat and is sitting on the grass	
A man is sitting in a field	A man with a backwards hat is sitting on the ground
A man with a backwards hat is sitting on the ground	A man is sitting in a field
There is no person wearing a hat and sitting on the grass	A man is sitting in the woods
Query: The blond girl is dancing behind the sound equipment	
The blond girl is dancing in front of the sound equipment	The blond girl is dancing in front of the sound equipment
The equipment in front of the blond dancing girl is sound	The equipment in front of the blond dancing girl is soundn
A little girl is looking at a woman in costume	Some teenage girls are dancing for the camera
Query: A large group of Asian people is eating at a restaurant	
A group of people in a large Asian restaurant is eating	A group of people in a large Asian restaurant is eating
A group of people from Asia is eating at a restaurant	A group of people from Asia is eating at a restaurant
A small group of people is waiting to eat in a restaurant	A small group of people is waiting to eat in a restaurant
Query: A large duck is flying over a rocky stream	
A duck, which is large, is flying over a rocky stream	A duck, which is large, is flying over a rocky stream
A seabird is flying over some rocks	A large white crane is flying near the water
A large white crane is flying near the water	A brown duck with a green head is flapping its wings in the water

Table 5.6: The retrieval cases using representations produced by BinSBERT and SBERT.

Chapter 6

Conclusions

In this thesis, different binarization approaches were compared in the context of sentence semantic matching, and an end-to-end architecture BinSiamese for learning binary representations was proposed.

We reproduced several binarization approaches, including three hard threshold-based methods, semantic-preserving autoencoder and C&D algorithm. We compared these methods from matching performance, memory footprint and inference latency perspectives. We found that methods perform variously on the downstream task when using different continuous representations. The dimension of binary representations corresponding to the best-performance is also different. We also observed that the binarized SBERT's representations could outperform the original continuous representations in matching performance on the MRPC dataset, even using the simple direct binarization method. We believe binarization brings this improvement by reducing learning noise. Besides, hard-threshold based approaches and semantic-preserving autoencoder perform relatively well on both datasets and are very practical methods. In contrast, due to the extremely high inference latency, the C&D algorithm is hard to be applied in real-world applications.

Inspired by semantic-preserving loss and the autoencoder, we proposed BinSiamese architecture to learn binary representations end-to-end and construct BinLSTM and BinSBERT using LSTM encoder and BERT encoder. Our proposed models achieve the best matching performance on both datasets. The matching performance of BinLSTM degrades only 6.6% on the SICK dataset and keeps the same performance of continuous representations on the MRPC dataset while compressing the memory footprint by 16 times. BinSBERT preserves 81% matching performance on the SICK dataset and improves 3% accuracy on the MRPC dataset, reducing the original memory footprint four times. Besides, our models are also competitive from the matching latency perspective.

We also performed an ablation study to investigate the effect of dimension of binary representations and semantic-preserving loss. We observed that our models are less sensitive to the dimension on the SICK dataset than on the MRPC dataset. We also found that the matching performance first increases and then decreases as the weight λ_{sp} of semantic-preserving loss increases, and when the λ_{sp} is large, the training curve fluctuates badly. Thus, λ_{sp} should be chosen carefully.

Semantic-preserving loss can cause fluctuation during training, a possible solution is to construct a continuous upper bound and optimize it. Norouzi et al [47] found a continuous upper bound of the Hamming distance-based triplet loss. Based on this research, future work can be done to figure out a reasonable continuous upper bound of semantic-preserving loss.

Bibliography

[1] Seonhoon Kim, Inho Kang, and Nojun Kwak. Semantic sentence matching with densely-connected recurrent and co-attentive information. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6586–6593, 2019.

[2] Weiwei Hu, Anhong Dang, and Ying Tan. A survey of state-of-the-art short text matching algorithms. In *International Conference on Data Mining and Big Data*, pages 211–219. Springer, 2019.

[3] Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, and Xueqi Cheng. A deep architecture for semantic matching with multiple positional sentence representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[4] Hang Li and Jun Xu. Semantic matching in search. *Foundations and Trends in Information retrieval*, 7(5):343–469, 2014.

[5] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*, 2015.

[6] Li Deng and Yang Liu. *Deep learning in natural language processing*. Springer, 2018.

[7] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

[8] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical*

Methods in Natural Language Processing, pages 670–680, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[9] Nils Reimers, Iryna Gurevych, Nils Reimers, Iryna Gurevych, Nandan Thakur, Nils Reimers, Johannes Daxenberger, Iryna Gurevych, Nils Reimers, Iryna Gurevych, et al. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019.

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

[11] Dinghan Shen, Pengyu Cheng, Dhanasekar Sundararaman, Xinyuan Zhang, Qian Yang, Meng Tang, Asli Celikyilmaz, and Lawrence Carin. Learning compressed sentence representations for on-device text processing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 107–116, 2019.

[12] Haohao Song, Dongsheng Zou, and Weijia Li. Learning discrete sentence representations via construction & decomposition. In *International Conference on Neural Information Processing*, pages 786–798. Springer, 2020.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4114–4122, 2016.

[14] Ting Chen, Martin Renqiang Min, and Yizhou Sun. Learning k-way d-dimensional discrete codes for compact embedding representations. In *International Conference on Machine Learning*, pages 854–863. PMLR, 2018.

[15] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *thirtieth AAAI conference on artificial intelligence*, 2016.

[16] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. A sick cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International*

Conference on Language Resources and Evaluation (LREC'14), pages 216–223, 2014.

[17] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[18] Hua He, Kevin Gimpel, and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1576–1586, 2015.

[19] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338, 2013.

[20] Wei Bao, Wugedele Bao, Jinhua Du, Yuanyuan Yang, and Xiaobing Zhao. Attentive siamese lstm network for semantic textual similarity measure. In *2018 International Conference on Asian Language Processing (IALP)*, pages 312–317. IEEE, 2018.

[21] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, pages 101–110, 2014.

[22] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. *Advances in neural information processing systems*, 27:2042–2050, 2014.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[25] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[26] Nouha Othman, Rim Faiz, and Kamel Smaïli. Manhattan siamese lstm for question retrieval in community question answering. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 661–677. Springer, 2019.

[27] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, 2020.

[28] Thanh-Toan Do, Tuan Hoang, Dang-Khoa Le Tan, Anh-Dzung Doan, and Ngai-Man Cheung. Compact hash code learning with binary deep neural network. *IEEE Transactions on Multimedia*, 22(4):992–1004, 2020.

[29] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[30] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th international conference on computer vision*, pages 2130–2137. IEEE, 2009.

[31] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[32] John Wieting and Kevin Gimpel. Paranmt-50m: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *ACL (1)*, 2018.

[33] Jamie Kiros and William Chan. InferLite: Simple universal sentence representations from natural language inference data. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4868–4874, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[34] Dinghan Shen, Qinliang Su, Paidamoyo Chapfuwa, Wenlin Wang, Guoyin Wang, Ricardo Henao, and Lawrence Carin. Nash: Toward end-to-end neural architecture for generative semantic hashing. In *Proceedings of the 56th Annual Meeting*

of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2041–2050, 2018.

[35] Bo Dai, Ruiqi Guo, Sanjiv Kumar, Niao He, and Le Song. Stochastic generative hashing. In *International Conference on Machine Learning*, pages 913–922. PMLR, 2017.

[36] Miguel A Carreira-Perpinán and Ramin Raziperchikolaei. Hashing with binary autoencoders. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 557–566, 2015.

[37] GE Hinton. Neural networks for machine learning, [video lectures], 2012.

[38] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[39] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *Twenty-eighth AAAI conference on artificial intelligence*, 2014.

[40] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3270–3278, 2015.

[41] Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. Deep learning of binary hash codes for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 27–35, 2015.

[42] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[43] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6309–6318, 2017.

[44] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[45] Julien Tissier, Christophe Gravier, and Amaury Habrard. Near-lossless binarization of word embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7104–7111, 2019.

[46] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

[47] Mohammad Norouzi, David J Fleet, and Russ R Salakhutdinov. Hamming distance metric learning. In *Advances in neural information processing systems*, pages 1061–1069, 2012.

Appendix A

Full Experiment Results

MODEL	PEARSON	SPEARMAN	LATENCY(μs)	NOTE
MaLSTM1024	0.824	0.757	103.42 ± 0.6	
HT-binary1024	0.595	0.499	103.9 ± 0.15	
Rand-binary1024	0.726 ± 0.006	0.649 ± 0.012	104.69 ± 0.54	
Rand-binary512	0.675 ± 0.026	0.586 ± 0.028	103.23 ± 0.12	
Rand-binary256	0.593 ± 0.046	0.503 ± 0.043	103.21 ± 0.18	
Rand-binary128	0.491 ± 0.04	0.412 ± 0.039	103.08 ± 0.18	
Rand-binary64	0.352 ± 0.09	0.298 ± 0.075	102.39 ± 0.15	
PCA-binary1024	0.619	0.616	110.59 ± 1.5	
PCA-binary512	0.631	0.648	108.26 ± 0.4	
PCA-binary256	0.651	0.662	106.78 ± 0.13	
PCA-binary128	0.674	0.669	106.3 ± 0.2	
PCA-binary64	0.669	0.651	105.92 ± 0.25	
C&D1024	0.366 ± 0.035	0.297 ± 0.026	61715.3 ± 6201.3	
C&D512	0.38 ± 0.028	0.336 ± 0.016	10912.2 ± 1109.7	
C&D256	0.371 ± 0.047	0.355 ± 0.031	2465.89 ± 440.32	
C&D128	0.32 ± 0.045	0.322 ± 0.037	710.5 ± 100.13	
C&D64	0.242 ± 0.108	0.228 ± 0.106	252.59 ± 18.27	
AE-binary-SP1024	0.759	0.7	105.63 ± 0.76	$\lambda = 0.2$
AE-binary-SP512	0.679	0.624	104.57 ± 1.8	$\lambda = 0.5$
AE-binary-SP256	0.674	0.599	103.36 ± 0.17	$\lambda = 0.0$
AE-binary-SP128	0.548	0.447	103.2 ± 0.16	$\lambda = 0.2$
AE-binary-SP64	0.502	0.412	103.2 ± 0.16	$\lambda = 1$

BinLSTM-Sp1024	0.77	0.706	106.43 ± 0.55	$\lambda = 0.1$
BinLSTM-Sp512	0.767	0.703	103.49 ± 0.23	$\lambda = 0.2$
BinLSTM-Sp256	0.785	0.714	102.85 ± 0.21	$\lambda = 0.1$
BinLSTM-Sp128	0.783	0.709	102.98 ± 0.17	$\lambda = 0.1$
BinLSTM-Sp64	0.763	0.679	102.47 ± 0.12	$\lambda = 0.1$
SBERT768	0.881	0.834	625.78 ± 2.89	
HT-binary768	0.861	0.828	611.75 ± 5.35	
Rand-binary768	0.858 ± 0.001	0.824 ± 0.002	614.37 ± 5.49	
Rand-binary384	0.848 ± 0.002	0.815 ± 0.002	611.97 ± 5.32	
Rand-binary192	0.831 ± 0.003	0.8 ± 0.004	611.27 ± 6.69	
Rand-binary96	0.796 ± 0.008	0.769 ± 0.006	609.37 ± 5.58	
Rand-binary48	0.743 ± 0.009	0.719 ± 0.009	610.68 ± 5.99	
Rand-binary24	0.66 ± 0.016	0.643 ± 0.015	610.89 ± 5.39	
PCA-binary768	0.67	0.723	619 ± 2.29	
PCA-binary384	0.718	0.746	615.52 ± 2.74	
PCA-binary192	0.746	0.754	615.83 ± 6.25	
PCA-binary96	0.768	0.757	612.85 ± 5.39	
PCA-binary48	0.772	0.75	611.43 ± 4.25	
PCA-binary24	0.747	0.716	609.52 ± 4.86	
C&D768	0.86 ± 0.001	0.827 ± 0.001	27726.7 ± 1351	
C&D384	0.849 ± 0.002	0.817 ± 0.002	5633.11 ± 451.09	
C&D192	0.828 ± 0.003	0.799 ± 0.003	1751.76 ± 132.95	
C&D96	0.791 ± 0.005	0.763 ± 0.005	926.29 ± 41.8	
C&D48	0.729 ± 0.011	0.708 ± 0.007	708.2 ± 10.43	
C&D24	0.644 ± 0.014	0.624 ± 0.01	647.8 ± 4.14	
AE-binary-SP768	0.857	0.813	617.45 ± 6.03	$\lambda = 0.8$
AE-binary-SP384	0.846	0.798	617.65 ± 5.78	$\lambda = 0.5$
AE-binary-SP192	0.833	0.786	616.89 ± 6.16	$\lambda = 0.2$
AE-binary-SP96	0.811	0.759	614.74 ± 4.59	$\lambda = 0.5$
AE-binary-SP48	0.756	0.698	614.27 ± 5.92	$\lambda = 0.5$
AE-binary-SP24	0.699	0.627	614.68 ± 6.45	$\lambda = 1$
BinSBERT-Sp768	0.866	0.815	618.21 ± 2.18	$\lambda = 1$
BinSBERT-Sp384	0.858	0.811	617.47 ± 1.47	$\lambda = 1$
BinSBERT-SP192	0.847	0.79	616.68 ± 0.97	$\lambda = 1$

BinSBERT-SP96	0.827	0.765	615.6 ± 3.73	$\lambda = 1$
BinSBERT-SP48	0.807	0.739	614.67 ± 1.33	$\lambda = 1$
BinSBERT-SP24	0.773	0.693	615.44 ± 1.36	$\lambda = 1$

Table A.1: The table shows the full experiment results on the SICK dataset. The number that follows the model name indicates the dimension of representations. For example, HT-binary1024 means the dimension of binary representations produced by HT-binary is 1024. The column NOTE record the hyperparameter λ of AE-binary-SP and BinSiamese models. MaLSTM and SBERT mean the real-value representations generated from MaLSTM and SBERT respectively. The rows following these two rows are the binarization approaches based on their representations or the pre-trained encoder.

MODEL	ACCURACY	LATENCY (μs)	NOTE
MaLSTM1024	0.711	102.39 ± 0.11	
HT-binary1024	0.665	102.97 ± 0.06	
Rand-binary-1024	0.69 ± 0.008	105.59 ± 4.84	
Rand-binary-512	0.684 ± 0.009	102.15 ± 0.16	
Rand-binary256	0.681 ± 0.008	101.78 ± 0.22	
Rand-binary128	0.672 ± 0.009	101.54 ± 0.19	
PCA-binary1024	0.685	108.44 ± 0.24	
PCA-binary512	0.692	106.14 ± 0.14	
PCA-binary256	0.69	104.99 ± 0.3	
PCA-binary128	0.688	104.11 ± 0.26	
C&D1024	0.681 ± 0.008	64823.7 ± 10002.39	
C&D512	0.676 ± 0.01	11298.22 ± 2217.52	
C&D256	0.676 ± 0.006	2249.23 ± 234.15	
C&D128	0.668 ± 0.006	723.64 ± 68.45	
AE-binary-SP1024	0.7	107.6 ± 0.23	$\lambda = 0.8$
AE-binary-SP512	0.696	104.34 ± 0.16	$\lambda = 0.5$
AE-binary-SP256	0.692	103.62 ± 0.43	$\lambda = 0.8$
AE-binary-SP128	0.689	102.93 ± 0.2	$\lambda = 0.2$
BinLSTM-Sp1024	0.697	107.78 ± 3.82	$\lambda = 0.5$
BinLSTM-Sp512	0.699	105.91 ± 3.97	$\lambda = 1$
BinLSTM-Sp256	0.711	105.09 ± 3.94	$\lambda = 0.8$

BinLSTM-Sp128	0.675	104.57 ± 3.91	$\lambda = 0.5$
SBERT768	0.743	2469.4 ± 5.99	
HT-binary768	0.759	2464.35 ± 5.74	
Rand-binary768	0.76 ± 0.008	2506.06 ± 6.25	
Rand-binary384	0.759 ± 0.008	2506.72 ± 4.29	
Rand-binary192	0.742 ± 0.006	2505.42 ± 6.47	
Rand-binary96	0.75 ± 0.006	2503.53 ± 4.98	
Rand-binary48	0.73 ± 0.009	2503.67 ± 8.89	
PCA-binary768	0.76	2508.23 ± 5.31	
PCA-binary384	0.76	2465 ± 6	
PCA-binary192	0.751	2461.01 ± 7.41	
PCA-binary96	0.74	2474.36 ± 8.54	
PCA-binary48	0.731	2445.07 ± 11.37	
C&D768	0.757 ± 0.005	30742.93 ± 2885.93	
C&D384	0.753 ± 0.009	7484.39 ± 384.47	
C&D192	0.755 ± 0.008	3562.72 ± 75.33	
C&D96	0.741 ± 0.011	2829.15 ± 19.14	
C&D48	0.731 ± 0.011	2611.64 ± 18.65	
AE-binary-SP768	0.766	2494.8 ± 5.58	$\lambda = 1$
AE-binary-SP384	0.76	2491.26 ± 5.39	$\lambda = 0.8$
AE-binary-SP192	0.758	2479.88 ± 6.76	$\lambda = 0.8$
AE-binary-SP96	0.741	2475.14 ± 6.28	$\lambda = 0.2$
AE-binary-SP48	0.743	2472.07 ± 6.88	$\lambda = 0.2$
BinSBERT-Sp768	0.773	2573.78 ± 2.68	$\lambda = 1$
BinSBERT-Sp384	0.744	2547.67 ± 3.48	$\lambda = 0.8$
BinSBERT-SP192	0.732	2510.15 ± 4.08	$\lambda = 0.8$
BinSBERT-SP96	0.725	2508.55 ± 1.57	$\lambda = 1$
BinSBERT-SP48	0.703	2501.75 ± 3.16	$\lambda = 0.8$

Table A.2: The table shows the full experiment results on the MRPC dataset. The number that follows the model name indicates the dimension of representations. MaLSTM and SBERT mean the real-value representations generated from MaLSTM and SBERT respectively. The rows following these two rows are the binarization approaches based on their representations or the pre-trained encoder.

Appendix B

Comments about the Speed of C&D

Our experiment results show C&D algorithm is not as fast as it said in the original paper. On the contrary, we find that the algorithm generates binary representations unacceptably slowly. We direct use the open-source code[1] from the author to conduct our experiments without any modification. The implementation is based on PyTorch. All matrics are sent to GPU by `matrix.cuda()` and all matrix operations are on the GPU. We treat the whole dataset as one batch rather than use the same batch size as it in the original paper, because our initial attempt shows using the small batch size is much slower, and the matching performance is worse either.

We notice in the original paper, it said Algorithm 1 only takes 4-6 iterations to converge. However, we observe it takes 3-20 iterations to converge, and the number of iterations is proportional to the dimension. This may be because we use the different continuous embeddings from the original paper. We think these more iterations cause the high latency we observed. We would also argue that the while loop in Algorithm 1 is the reason for the slow computing speed since it is not able to be optimized on GPU.

[1]https://github.com/songs18/cd_algorithm