The Decodable Sentence-BERT trained for VMware Info-Retrieval System

Abstract

Sentence-BERT[1], an expanded BERT-based[2] language model with additional architecture and training tasks, maintains state-of-the-art performance in semantic textual tasks (STS[3] and uSTS[4], etc.) with much lower computational cost compared to BERT or RoBERTa[5]. It trains the BERT's final representative with siamese and triplet network architecture and transfers it to semantic sentence embedding -- sentences can be aggregated and sorted by similarity functions. Sentence-BERT is commonly used in information retrieval(info-retrieval) system, for example, document searching system(vAQA[6]) or log aggregation system(NimbusEC[7]).

In this paper, we present the Decodable Sentence-BERT, a modification in both model architecture and training tasks to further extend the model's ability. We add Decoder Transformers stacks on top of the Sentence-BERT and derive the original sentences through the Decoder stacks from the last hidden states of Sentence-BERT. It not only keeps all the ability of the Sentence-BERT but also enables the model to recover the original sentence from the sentence embedding -- we can drop the original sentences after embedding. What's more, we can use the accuracy of recovering sentences to indicate the confidence of classification tasks during inference, which is more reliable than the traditional confidence we used. We further pre-train the model with VMware Documents which enables it to perform textual semantic tasks in VMware's info-retrieval system.

We evaluate the Decodable Sentence-BERT on common STS tasks, VMware documents, and Test Logs from a QE team in VMware. The model holds Sentence-BERT's performance in aggregation tasks, outperforms both BERT and Sentence-BERT at classification tasks by 22 points, and improves storage efficiency by 1.8 times.

1. Introduction

Encoder, Decoder, and Sequence-to-Sequence are the three main architectures for NLP(Natural Language Processing) relevant models. After the Transformer[8] was invented, it showed its powerful learning and transfer learning[9] ability and brought all three architectures to a new level.

As the LLM(Large Language Model) like GPT[10](Generative Pre-trained Transformers from OpenAI) and LLaMA[11](from Meta) set new state-of-the-art performance in the Decoder-Only domain for many generative semantic tasks, Encoder-Only mode is still capable for info-retrieval tasks with lower hardware requirements and costs.

Among all Encoder-Only-architecture models, BERT[2] (Bidirectional Encoder Representations from Transformers) was the state-of-the-art model for sentence-level semantic tasks like classification or similarity comparison. The capability to do semantic similarity and classification for BERT is designed from both the architecture and training tasks. BERT uses a sentence pair with an indicator token to indicate the relationship between the pair during the training. The indicator learns sentence-level semantic meanings and their relationship after the training.

One problem with this architecture is that we have to do N-by-N inferences if we want to compare N sentences with each other. Even if the BERT model can generate sentence embedding(the last hidden states of the indicator), as we don't train the embedding to be similar in any method, we cannot use the embedding for similarity-related tasks. This is commonly required for info-retrieval systems.

To reduce computational costs for semantic similarity tasks, the Sentence-BERT was invented(Nils Reimers, 2019). It can map each sentence into vector space which makes semantically-similar sentences close to each other spacially.

Sentence-BERT is a BERT-based model. It adds siamese[12] and triplet[13] neural network architecture with additional training tasks to further pre-train BERT. This practice reforms the BERT embeddings to be meaningful when doing semantic similarity tasks. The complexity for finding the most similar sentence pair in a collection of N sentences reduces from N^2 inferences to N inferences + N comparisons, which reduces the computational efforts significantly.

When developing the info-retrieval system, we save both sentence embeddings and the original sentences. The sentence embeddings, in the format of realed-number multi-dimensional vectors, usually take 1/4 of the storage compared to the original sentences. This expands the storage requirements for the info-retrieval system. Inspired by the Sequence-to-Sequence architecture which is commonly used in natural language translation (a source Encoder to encode the original language sentences and a target Decoder to decode the hidden states to the target language sentences), we further extend the model architecture by adding the Decoder stacks on top of the Sentence-BERT with training tasks. Different from the Sequence-to-Sequence translation model, we "translate" the source sentence to itself. We use all BERT and Sentence-BERT training materials and tasks to train the Decodable Sentence-BERT. As a result, the model holds similar performance compared with the Sentence-BERT with the extra ability to recover the sentence from sentence embedding. After fine-tuning, around 70% of the storage can be saved by storing only the sentence embeddings with a correction patch from the model, which leads to a 3x storage efficiency.

The new model's architecture brings another benefit. During sequence classification tasks, we can ask the model to generate both the classification result and the original sentence. The accuracy of the recovering sentence can indicate the accuracy of the classification result(Chapter 3.). This outperforms the "inference confidence"(the weight balance between all classifications) by 22 points.

The paper is structured in the following Chapters: Chapter 2 describes all related models; Chapter 3 presents the Decodable Sentence-BERT in detail with training tasks; Chapter 4 evaluates the model on VMware Datasets.

2. Related Works

The Decodable Sentence-BERT is built on top of many models. In this chapter, we will go through all fundamental models in sequence in detail. After the base model is described, I will focus more on explaining the improvements.

2.1 Transformer

Transformer[8] was a new architecture that replaced the recurrence method(RNN[14] for example) by the parallel method and leverage attention mechanism[15] to the new extents proposed by Vaswani et, al in 2017.

Instead of the recurrent neural network models(LSTM[16], etc) that keep fitting tokens to the same model in sequence, leverages and activates the previous hidden states of the model to generate new states, Transformer fits all tokens at once with a positional encoding function to help identifying sequence relationship.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \quad (1)$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \quad (2)$$

This accelerate both training and inference computational speed and reduce the computational complexity from O(N) to O(1) for each sample -- the change makes the model to be more efficient and capable to larger datasets, which expands the model ability to a new level.

What's more, the Transformers introduced another critical change. Inspired by the retrieval system, they merged the concept of Query, Key and Value with the joint attention mechanism and proposed a multi-heads self-attention architecture. The Query and Key product weight maps that adjust the "attention" applied on the Value. After summing up all weighted Value with some additional operations, a new representative is generated to be the output of a Transformer Layer.

$$(Q,K,V) = ReLU(W^{(q,k,v)} I^T) \quad (3)$$

$$Attention(Q,K,V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (4)$$

(I: input embeddings, W: learnable weight matrix)

The semantic ability are increased by stacking the Transformer Layers on top of other. Each Transformer Layer receives the output of the previous Transformer Layer and generate it's own representative.

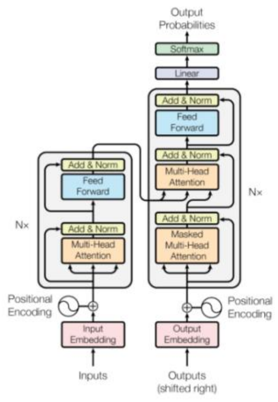

Figure 1. Encoder(left) and Decoder(right) Architecture[8]

The paper proposed two architectures: the Encoder(Diagram 1a) and the Decoder(Diagram 1b). The Decoder has a similar architecture to the Encoder with only one difference -- the Decoder has two Multi-Head Attentions Layers. It stacks Decoder -To-Encoder attention on top of the self-attention part.

The model gets different abilities by using different components. The Encoder-Only architecture generates new embeddings from the source, which makes it suitable for semantic comparison and classification tasks. Models like BERT[2] or RoBERTa[5] are the successor of this architecture. The Decoder-Only architecture can transform hidden states into the source, which makes it capable of generative tasks. GPT[10](from OpenAI) or LLaMA[11](from Meta) set state-of-the-art performance in this area. The Encoder-Decoder architecture, also known as the Sequence-to-Sequence architecture, can transform the source(text) to another source(picture), which is also commonly used worldwide.

One important feature of the Transformer is transfer learning. After pre-training each component properly, only a few efforts are required to connect different components or the same component with different tasks. For example, we can train a stack of Encoder-Only Transformers on voice and Decoder-Only Transformers on text separately and connect them afterward. After only several rounds of fine-tuning, the model can be capable of performing voice-to-text tasks.

2.2 BERT

In 2018, Jacob et al. proposed some new pre-training methods for Encoder-Only Transformers which set state-of-the-art performance in semantic textual tasks. BERT, which stands for the Bidirectionalnc Eoder Representations from Transformers, includes a new representation and two new training task, makes the model capable for several down-stream tasks that requires only minimum efforts of fine-tuning.

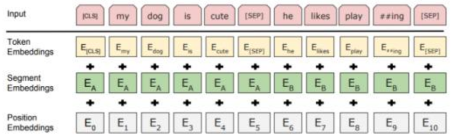
Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2. BERT input representation

They proposed a new representation of the inputs: [CLS] sentence 1 [SEP] sentence 2* [SEP]*(*represents to be optional). [CLS] is an indicator that represents the "classification" and [SEP] represents the "separation" that isolates two sentences. They also proposed a new token [MASK] which was used in their new training tasks.

What makes BERT outstanding is the two new training tasks, the Masked Language-Model Task and the Next-Sentence Prediction Task. The two training tasks are trained jointly during the pre-training phase. They use the BooksCorpus [17](800M words) and English Wikipedia (2500M words) to pre-train the BERT.

The Masked Language-Model(MLM) Task

In the pre-training phase, they randomly masked some tokens from the original sentences and asked the model to predict those tokens. Concretely, they "mask" the original sentences in two ways: they replace the original words with the special token [MASK] and set attention black hole for those masks -- they simply set a huge negative number (-1e9) to block the self-attention mechanism applied on those tokens. They replace words with random words in the vocabulary and they don't interfere with the attentions in this case. These two methods force the model to learn word-level relationships -- the model needs to predict the masked words and correct the faulty words by looking at other words. This method helps the model to learn word embedding.

One more detail for the Masked LM Task is that the special token [CLS] will never be masked so that the model can always rely on it when predicting masked tokens. This helps the model to generate sentence-level embedding in the [CLS] embedding.

$$Loss_{mlm} = \sum_{i=1}^{n} CrossEntropy(W_{(d_{vocab},d_{model})}(H^{(i)}_{[attn,:]})^T, Y^{(i)}_{[attn]}) \quad (5)$$

(H: last representitive, Y: tokens, n: sample numbers, attn: not padded tokens)

The Next-Sentence Prediction(NSP) Task

During the Next Sentence Prediction Training task, they fill the model with two sentences and use the [CLS] token to be the indicator for the relationship between them: if the second sentence entails the first sentence, the model should predict IsNext(1) otherwise NotNext(0) from the [CLS] indicator. When building the training datasets, they keep 50% of the sentences in the original sequence and mark them as IsNext and randomly shuffle the other 50% of the sentences and mark them as NotNext. This method forces the model to learn sentence-level embedding -- the model can make the inference only after it captures the semantic meaning from the two sentences.

The key point that makes BERT easy to fine-tune for a wide range of downstream tasks is the [CLS] token. As the model learns sentence meaning from this indicator, we can further build a classification/regression network on top of this token.

$$Loss_{nsp} = \sum_{i=1}^{n} CrossEntropy(W_{(2,d_{model})}(H_{[0,:]}^{(i)})^T, \{0,1\}^{(i)}) \quad (6)$$

2.3 Sentence-BERT

As mentioned in the Introduction, the Sentence-BERT[1](Nils et al. 2019) reduces the inference complexity from $O(N^2)$ to $O(N)$ for sentence semantic comparison tasks while it can still maintain the performance of other semantic textual tasks(classification, etc.) compared to BERT.

Sentence-BERT is a BERT-based pre-training method applied with additional structure. The method forms the BERT representative to be comparable in the multi-dimensional vector space by similarity functions -- unlike we need to feed the BERT model with both target sentences to be compared, we transform sentences to the Sentence-BERT representatives first and then make the comparison afterward.

Average-Pooling

In Chapter 2.2, I described the use of the indicator token [CLS] of BERT. BERT uses the last representative of [CLS] for most of downstream tasks. We can call it CLS-Pooling as we choose the [CLS] representative among BERT's last hidden states.

$$\hat{W}_{(1,m)} = F_{pooling}(W_{(n,m)}) \quad (7)$$

For Sentence-BERT, they use the Average-Pooling or Mean-Pooling for their training tasks -- they found it outperforms CLS-Pooling during model evaluation.

Model	**MR**	**CR**	**SUBJ**	**MPQA**	**TREC**	**MRPC**	**Avg.**
BERT CLS-vector	78.68	84.85	94.21	88.23	91.4	71.13	84.66
SBERT-NLI-base	**83.64**	**89.43**	**94.39**	**89.86**	**89.6**	**76**	**87.41**

Table 1. Sentence-BERT Mean-Pooling vs. BERT CLS-Pooling[1]

2.3.1 Pre-Training Funtions

They propose three pre-training functions.

Classification Objective Function. They concatenate the sentence embeddings u and v with the element-wise difference$|u - v|$and multiply it with a Linear Layer.

$$loss_{cls} = CrossEntropy(W_{(3n,d_{label})}^{T}(u, v, |u - v|)) \quad (8)$$

(n: the dimension of the sentence embeddings)

Regression Objective Function. They use MSE(Mean-Squared Error) Loss with cosine-similarity function for the sentences u, v.

$$loss_{reg} = MSE(\cos(u, v), l), l \in R_{(-1,1)} \quad (9)$$

Triplet Objective Function. Given an anchor sentence S_a, a positive sentence S_p, and a negative sentence S_n, triplet loss tunes the network to make the distance between S_a and S_p is smaller than the distance between S_a and S_n. Mathematically, they minimize the following loss function.

$$loss_{tri} = Max(\|S_a - S_p\| - \|S_a - S_n\| + \varepsilon, 0) \quad (10)$$

They train the model on the combination of the SNLI[18] (Bowman et al., 2015) and the MNLI[19] (Williams et al., 2018) datasets. It contains 1M sentence pairs annotated with "contradiction", "entailment" and "neutral" and a range of genres of spoken and written texts. They use Classification Objective Function with one epoch to train the model.

3. The Decodable Sentence-BERT

In this Chapter, I will split Decodable Sentence-BERT into two parts and introduce them seperately.

3.1 The Model

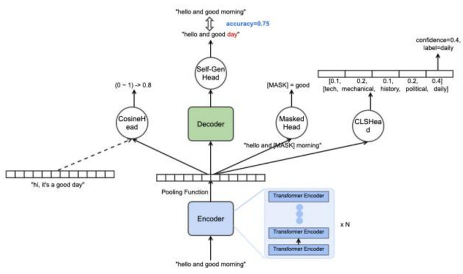

Figure 2. the Decodable Sentence-BERT

The Decodable Sentence-BERT is a Sequence-To-Sequence(Encoder-To-Decoder) architecture Model(Chapter 2.1). We choose the Sentence-BERT to be the Encoder part, and stacks several standard Decoder Transformer on top of it to be the Decoder part.

We reuse the BERT input format(Chapter 2.2) on the Decoders with slightly different meanings. We use [CLS] to represent the start signal(indicator) of a sentence and [SEP] to represent the end of a sentence. This makes it different from BERT -- unlike the BERT that can take two sentences isolated by [SEP], the Decodable Sentence-BERT can only take one.

The design of the Encoder parts follows the Sentence-BERT. We use Average-Pooling function(Chapter 2.3.1) that transforms the last hidden states to be the final representative of the Encoder part.

Similar to the Encoder part(Figure 1.), we create an Embedding Layer and an Extraction Layer for Decoder part, tie weights between them[20]. We cannot share the Embedding Layer between the Encoder part and the Decoder part. Apart from the different use of the indicator token [CLS] and [SEP], the Embedding Layer from the Decoder part is used to predict next token from the sentences(Figure 1.) but the Embedding Layer from the Encoder part is used to predict what it absorbs with masks.

More specifically, We keep the model to offer common and fundamental functions and we split and separate different training and inference tasks with their specific structures from the model and name them "Heads". This architecture makes the Decodable Sentence-BERT to be an All-In-One model -- the model is adaptable and capable for different kind of tasks with different combination of the Model and the Heads. What's more, different Heads can be loaded and saved separately which lead to lower GPU memory consumption.

Components	**Size(MB)**	**Parameters**
Encoder Decoder	216.40	Heads:12, Head_Dimension:64 Hidden_Size:768,Feedforward_Size: 3072, Layers:6
Encoder Embedder(Extractor) Decoder Embedder(Extractor)	90.92	Vocab_Size: 30522
Self-Generative Head Masked-LM Head	91.68	Hidden_Size: 768, Linears: 2 Feedforward_Size:3072
Classification Head (fine-tuning only)	28.33	Hidden_Size 768,Linear: 2 Feedforward_Size: 3072
Cosine-MSE Head	0	/

Table 2. Model Hyper-Parameters

During pre-training, we use Masked LM Objective function(Chapter 2.2) and Cosine Regression function(Chapter 2.3.2) for the Encoder part and Self-Generative function(3.2) for the Decoder part. We use the BooksCorpus (800M words) and English Wikipedia (2400M words) with MLM Head and SG Head to train the model first, and then feed it with the mixture of SNLI, MNLI, STS and Quora Duplicate Questions[22] with MLM, C-MSE and SG Heads.

Combination	Dataset	Loss Functions
Encoder+MLM Head &Decoder+SG Head	BookCorpus English Wikipedia vBERT Documents	CrossEntropy
Encoder+MLM Head & Encoder+C-MSE Head & Decoder + SG Head	SNLI,MNLI, STS, QDQ	CrossEntropy, MSE

Table 3. Pre-Training datasets and loss functions

We fine-tune the model with VMware Documents[21]. As estimated, it keeps all semantic textual ability, outperform the basic BERT or RoBERTa Model in VMware specific tasks with extra ability to recover sentences from sentence embeddings with high accuracy. Detail will be displayed in the Experiment Chapter(4.).

3.2 The Heads

We define four Heads with four different tasks. In pre-training phase, we use the Masked-LM Head to derive sementic representative from sentences. We use the Cosine-MSE head to form sentence representative to be meaningful spacially. The Self-Generative Head is jointly trained with Masked-LM Head and Cosine-MSE Head in pre-training Phase and also involved in fine-tuning. We use Sequence Classification Head to fine-tune the model on specific classification tasks. All components including the model are unique apart from SE Head -- it varies in different use cases so we save multiple SC Heads for different uses.

The Masked-LM Head

The structure of the Masked-LM head is the same as BERT's Masked LM Task's structure. It reuses the Embedding Layer of Encoder, transpose it to be the Extractor of embeddings[20].

$$W\_ext_{(d_{model}, d_{vocab})} = W\_emb_{(d_{vocab}, d_{model})}^{T} \tag{11}$$

During pre-training, 15% of the tokens from the source sentence are masked as [MASK] and another 15% of the tokens are replaced by a random token from the vocabulary[23].

$$Loss_{msk} = \sum_{i=1}^{n} CrossEntropy(W_{(d_{vocab}, d_{model})}(H_{[masked,:]}^{(i)})^{T}, Y_{[masked]}^{(i)}) \tag{12}$$

$$Loss_{mlm} = Loss_{msk} + Loss_{attn} \tag{13}$$

We made a difference when calculating loss. Apart from the standard loss(5), we make another comparison only on those tokens that are masked(12). We sum-up both loss to be the final loss(13) -- this emphasis the model to make correct prediction on masked tokens, which accelerates the learning phase.

The Cosine-MSE Head

As Sentence-BERT proposes three training objectives that help form the final representative of BERT to be spacially meaningful, we choose Siamese network with Cosine similarity function with MSE Loss and build it as a Head(9). We perform Cosine-Similarity training after the basic BERT pre-training phase with the mixture of SNLI, MNLI and QDQ(Table.2). The model has a slight performance drop on the semantic similarity comparison tasks compared to the Sentence-BERT.

The Self-Generative Head

The objective of the Self-Generative Head is to derive source sentences from the Decoder part's outputs. The structure of the Self-Generative Head is almost the same as the Masked-LM head -- the Masked-LM head accepts the last hidden states from the Encoder (no pooling) and predicts the masked source sentence using transposed Encoder Embedding Layer, while the Self-Generative Head accepts the last hidden states from the Decoder and predict the source sentence using transposed Decoder Embedding Layer. In Self-Generative Head, we don't apply any masks on the source sentence.

The Sequence Classification Head

The Sequence Classification Head has basically the same structure as the NSP task(Chapter 2.2.)(6) with three differences. First, we use Average-Pooling(Chapter 2.3.)(7) rather than CLS-Pooling. Second, the number of the labels varies based on use cases -- the NSP task can be considered as a binary(2-label) classification task. Last but not least, it can only take one sentence(Chapter 3.1). The Sequence Classification will not be pre-trained.

3.3 Training

During training, each sample(sentence) goes through both the Encoder Embedding Layer and the Decoder Embedding Layer. The Decoder part takes the pooled hidden states(Chapter 2.3.1.) from the Encoder parts and the output from the Decoder Embedding Layer(Chapter 3.1.). As mentioned in Transformers(2.1), each Decoder Transformer has two attention layers(Figure 1.). The Decoder Transformer's self-attention layer generates a new representative based on the previous Decoder Transformer's output. If the Decoder Transformer is the first layer, it takes the output of the Decoder Embedding Layer. The Encoder-Decoder attention layer, takes both the last hidden states of the Encoder part and the previous Decoder Transformer's output.

Another thing to note is, we make a shift between the target sentence and the source sentence. We use [CLS]+source_tokens[0]+source_tokens[1]+...+source_tokens[-1] to be the source sentence and the target_tokens[0]+target_tokens[1]+...+target_tokens[-1] +[SEP] (source_tokens=target_tokens) to be the target sentence. The decoder needs to make the next-one prediction for each token, even though this happens in parallel during training(the prediction is generated at once). What's more, we need to apply a future-visible attention mask instead of random masks(for Masked-LM) during training. The future-visible attention mask is a diagonal matrix in which only the tokens at rights can see tokens at lefts, not vice versa. This forces the model to predict tokens at rights(in the future) by consuming the meaning from tokens at lefts(previously), which simulates the situation during inference -- it generates new tokens by consuming what it has generated.

3.4 Inference

Using the Self-Generative Head for inference is different from training. Instead of predicting sentence at once when training, the prediction repeats several times. During inference, only [CLS] token will be sent to the Decoder and Self-Generative Head first. The Decoder part and the Self-Generative Head then generates one token(predicted_tokens[0]) by taking the [CLS] embedding and the pooled embedding(Chapter 3.3.). In next round, we fit the model with [CLS]+predicted_tokens[0] and achieves a predictions of predicted_tokens[1]. This repeats until either the [SEP] token(the indicator of EOL, Chapter 3.1.) is generated or the predicted_tokens met the length restriction. As this behavior keep generating new tokens by consuming what the model generated, people name it as "Self-Generative".

4. Experiments

Dataset	Description	Average Length	Samples	Labels
QE Test Logs 1	Unit Test Log Piece(complete)	114 words	25803	8
QE Test Logs 2	Regression Log Pieces extracted from a series of organized test operations	239 words	760	\

Table 4. Dataset used for evaluation

We did two experiments to evaluate the semantic performance of the model. We also recorded the recovering

name	type	512 words	768 dimensions	512 dimensions	384 dimensions
sentence	text(string)	3844(Bytes)	\	\	\
vector_32	32bits float	\	3168 (82%)	2144(56%)	1632(42%)
vector_16	16bits float	\	1585(41%)	1072(28%)	816(21%)

Table 5. sentence and vector size comparison

accuracy and the final storage enhancements for each experiment. For QE Test Logs 1, we splited the datasets to be train/eval and test set. The Train/Eval set were from the same data distribution(same month) and the Testset was from a different data distribution(next month).

Aggregation

Dataset	Samples	Threshold	Sentence-BERT	The Decodable Sentence-BERT
QE Test Logs 1	25803	0.95	1079 clusters	**964 clusters**
QE Test Logs 2	760	0.78	358 clusters	**331 clusters**

Table 6. Aggregation Semantic Performance

As displayed in Table 6., we used the DBSCAN with Cosine Similarity to aggregate logs. The Decodeable Sentence-BERT has a slight advance in the performance comparing to the Sentence-BERT(MiniLM-v6).

Classification

model	Layers	Epoch	EvalSet Accuracy classification(Self-Gen)	TestSet Accuracy (high confidence)	Compression ratio
Distilled BERT	6	6	0.97(\)	0.59	\
Sentence-BERT	6	6	0.95(\)	0.55	\
Decodable Sentence-BERT	**6**	**6**	**0.95(0.92)**	**0.81(0.79)**	**65%**

Table 7. Classification Semantic Performance

We used the QE Test logs 1 to be the classification dataset. All models are fine-tuned for 6 epochs and we selected the best-performance checkpoints. For Evalset, we recorded the classification accuracy for all models; we recorded the self-generative accuracy for Decodable Sentence-BERT. For Testset, we set 0.9 to be the BERT and Sentence-BERT weighted confidence threshold(Figure 2.) and 0.8 to be the Decodeable Sentence-BERT self-generative accuracy confidence threshold. When computing compression ratio, we used the testset self-generative accuracy with a correction patch which fixed the mis-predicted words to be the storage cost(14).

$$Ratio = 1 - (Size_{vector} + Size_{sentence \cdot (1-acc)}) \;/\; (Size_{sentence}) \quad (14)$$

5. Conclusion

In this paper, we proposed a new Sequence-To-Sequence model called the Decodable Sentence-BERT. We walked through all the fundamental algorithms and formulas in Chapter 2 and explained the model in Chapter 3. In Chapter 4, we made a comparison of the model with the Sentence-BERT and BERT on several info-retrieval scenarios of VMware and proved the model to be superior. The model is capable of performing all BERT-based or Sentence-BERT-based semantic tasks for VMware.

6. Future Work

BPE, as known as Byte-Pair Encoding, is an algorithm to encode strings of text into tabular form. It is widely used in the large language model(LLM) tokenizer to expand the model vocabulary. We plan to use BPE to train another model to handle numbers, special tokens and special formats(JSON, etc.)

What's more, we plan to train and evaluate three versions of the model. Mini model contains 6 layers and 12 attention heads for the Encoder and the Decoder. Base model contains 9 layers and 12 attention heads and Large model contains 12 layers and 24 attention heads.

Last but not least, we plan to collect more document material from VMware including logs from Support Bundles. This will enable the model to be capable for Support Bundle relevant tasks.

7. References

[1] Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Nils Reimers and Iryna Gurevych. https://arxiv.org/pdf/1908.10084.pdf

[2] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. https://arxiv.org/pdf/1810.04805.pdf

[3] SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Cross-lingual Focused Evaluation. Daniel Cera , Mona Diab, et al. https://aclanthology.org/S17-2001.pdf

[4] Unsupervised Sentence Textual Similarity with Compositional Phrase Semantics. Zihao Wang, Jiaheng Dou, et al. https://arxiv.org/pdf/2210.02284.pdf

[5] RoBERTa: A Robustly Optimized BERT Pretraining Approach. Yinhan Liu, Myle Ott, Naman Goyal et al. https://arxiv.org/pdf/1907.11692.pdf

[6] vAQA: RAIL/vAQA+Service+Architecture

[7] NimbusEC: DevToolsDocKB/Nimbus+Error+Classification

[8] Transformers: Attention Is All You Need. Illia Polosukhin, et al. https://arxiv.org/pdf/1706.03762.pdf

[9] Transfer learning: a friendly introduction. Asmaul Hosna, Ethel Merry, et al. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00652-w

[10] Improving Language Understanding by Generative Pre-Training. Alec Radford et al. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[11] LLaMA: Open and Efficient Foundation Language Models. Hugo, el al. https://arxiv.org/pdf/2302.13971.pdf

[12] Siamese neural networks: an overview. Chicco, et al. Artificial Neural Networks, Methods in MolecularBiology. https://link.springer.com/protocol/10.1007/978-1-0716-0826-5_3

[13] Deep metric learning using Triplet network. Elad Hoffer, Nir Ailon, 2014. https://arxiv.org/pdf/1412.6622.pdf

[14] Recurrent Neural Networks (RNNs): A gentle Introduction and Overview. Robin M. Schmidt, 2019. https://arxiv.org/pdf/1912.05911.pdf

[15] Neural Machine Translation by Jointly Learning to Align and Translate. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, 2014. https://arxiv.org/pdf/1409.0473.pdf

[16] LSTM: Long Short-Term Memory-Networks for Machine Reading. Jianpeng Cheng, Li Dong, and Mirella Lapata. https://arxiv.org/pdf/1601.06733.pdf

[17] BookCorpus: Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. Yukun Zhu, Ryan Kiros et al. , 2015. https://arxiv.org/pdf/1506.06724.pdf

[18] SNLI: A large annotated corpus for learning natural language inference. Samuel R. Bowman et al. https://arxiv.org/pdf/1508.05326

[19] MNLI: A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. Adina Williams et al. https://arxiv.org/pdf/1704.05426.pdf

[20] Using the Output Embedding to Improve Language Models. Ofir, et al. https://arxiv.org/pdf/1608.05859.pdf

[21] vBERT: a pre-trained BERT for VMware https://confluence.eng.vmware.com/display/RAIL/vBERT

[22] Quora.https://www.kaggle.com/c/quora-question-pairs