A SEB ALGORITHM

Algorithm 1 illustrates the detailed steps of our proposed subword to byte sequence mapping in Section 4.1. Algorithm 2 gives the step-by-step operations to obtain the subword embedding with SEB based on bytes, given the mapping in Algorithm 1.

Algorithm 1: Construction of Subword to byte sequence mapping

Input : Byte vocabulary $\mathcal{V}_b = \{0, 1, \ldots, V_b - 1\}$;
Subword vocabulary $\mathcal{V}_w = \{w_0, w_1, \ldots, w_{V_w-1}\}$;
The number of bytes per subword n
Output : Mapping: $\mathcal{M} : \mathcal{V}_w \rightarrow (\mathcal{V}_b)^n$

for $w_i \in D_w$ **do**
  repeat
    for $j \leftarrow 1$ *to* n **do**
      Sample $b_{ij} \sim$ Uniform[0, V_b) // $\mathrm{P}(b_{ij}) = \frac{1}{V_b}, \quad b_{ij} \in \mathcal{V}_b$
    end
  until $(b_{i1}, b_{i2}, \ldots, b_{in}) \notin \mathcal{M}$
  Add $w_i \rightarrow (b_{i1}, b_{i2}, \ldots, b_{in})$ to $\mathcal{M}$
end
return $\mathcal{M}$

Algorithm 2: Subword embedding with SEB

Input : Subword to byte sequence mapping $\mathcal{M} : \mathcal{V}_w \rightarrow (\mathcal{V}_b)^n$;
Subword sequence $S = (w_1, w_2, \ldots, w_m)$;
A feed-forward network FFN;
Embedding matrix is $\mathbf{B} \in \mathbb{R}^{V_b \times d}$;
Subword embedding dimension d'
Output : Subword embeddings $\mathbf{E}' \in \mathbb{R}^{m \times d'}$

for $i \leftarrow 1$ *to* m **do**
  $(b_{i1}, b_{i2}, \ldots, b_{in}) \leftarrow \mathcal{M}[w_i]$
end
Byte sequence for $S : (b_{11}, \ldots, b_{1n}, \ldots, b_{m1}, \ldots, b_{mn})$
$\mathbf{E} \in \mathbb{R}^{mn \times d} \leftarrow$ retrieve $(b_{11}, \ldots, b_{1n}, \ldots, b_{m1}, \ldots, b_{mn})$ from $\mathbf{B}$
$\tilde{\mathbf{E}} \in \mathbb{R}^{m \times nd} \leftarrow$ reshape $\mathbf{E}$ (in a row-major order)
$\mathbf{E}' = \mathrm{FFN}(\tilde{\mathbf{E}}) \in \mathbb{R}^{m \times d'}$
return $\mathbf{E}'$

B EXPERIMENTAL DETAILS AND MORE RESULTS

B.1 ENVIORMENTAL SETTINGS

All the programs in our work are implemented using Python 3.9.0, Fairseq Ott et al. (2019), PyTorch 1.13.0, and CUDA 11.7. For the hardware environment, we run all codes on a machine with Intel i7-11700K CPU, 64G memory, and NVIDIA GeForce RTX 3080 GPU.

B.2 PREPROCESSING DETAILS

Translation For IWSLT14, there are 166K sentence pairs for training and validation and 5.6K for testing. The vocabulary shared by the source and target languages is built by BPE Sennrich et al. (2015) with 10K tokens.

For WMT14, en-de contains 4.5M sentence pairs. Newstest2013 is used for validation and newstest2014 for testing respectively. The merge operation is 32K for BPE and the dictionary is shared by source and target.

Sentiment analysis There are 25000 training samples and 25000 testing samples for IMDb. We take 25% of the training data for validation and the rest for training. For SST2, The training, validation, and test examples in SST2 are 67349, 872, and 1821, respectively. The tokenizer is "basic_english" in the TorchText package. The minimum frequency needed to include a token in the vocabulary is 5. The maximum length of the sentence is 256.

B.3 Implementation details

Translation The baseline we compare is the transformer with subword embedding Vaswani et al. (2017). Our proposed method only replaces the subword embedding with SEB.

For IWSLT14, the encoder and decoder layers are both 6 and have 4 attention heads. The hidden dimension of attention is 512 and the dimension of the feedforward layer is 1024. The optimizer is Adam Kingma & Ba (2014) with an inverse square root learning rate scheduler, and warm up 4000 steps. The learning rate is 5×10^{-4}. The total training epochs are 100, and we average the best 5 checkpoints for testing.

For WMT14, the encoder and decoder layers are both 6 and have 8 attention heads. The hidden dimension of attention is 512 and the dimension of the feedforward layer is 2048. The optimizer is Adam Kingma & Ba (2014) with an inverse square root learning rate scheduler, and warm up 4000 steps. The learning rate is 5×10^{-4}. The total training epochs are 100 and we use the early stop if the validation loss does not decrease in 5 epochs. We average the best 5 checkpoints for testing.

Sentiment Analysis We use 2-layer BiLSTMs for both IMDb and SST2 classification tasks. We keep all model architectures the same for the baseline models and models with our SEB$_{co}$ except for the embedding parts. The subword embedding dimension is 64 and 256 for IMDb and SST2. The hidden units are 64 and 300 for IMDb and SST2. The hidden dimension of 2-layer FFN in SEB$_{co}$ is 128 for both datasets. We optimize the model using Adam Kingma & Ba (2014) and the learning rate is 5×10^{-4} for the baseline and our method on both datasets. The best model parameters evaluated on validation data are applied for testing.

Language modeling In this experiment, we also use a two-layer FFN in SEB, which has 4096 hidden units. The architecture is `transformer_lm` in the Fairseq framework. We share the input and output embedding in the encoder and the other hyperparameters and settings are the same as Fairseq.

B.4 Analysis for semantic meaning

In this section, we will analyze whether the derived subword embeddings from our method can truly encode the meaning of the words in the embedding space. To experiment on this aspect, we mainly calculate the cosine similarity between two word embeddings obtained based on our method SEB. We list some examples in IMDb sentiment analysis in Table 7.

	good	**great**	**funny**	**bad**	**worse**	**boring**
good	1	0.63	0.49	-0.58	-0.61	-0.58
great	0.63	1	0.40	-0.53	-0.33	-0.38
funny	0.49	0.40	1	-0.72	-0.61	-0.60
bad	-0.58	-0.53	-0.72	1	0.72	0.85
worse	-0.61	-0.33	-0.61	0.72	1	0.88
boring	-0.58	-0.38	-0.60	0.85	0.88	1

Table 7: The cosine similarity of the subword embeddings calculated based on SEB.

The cosine similarity demonstrates that the subword embedding of our proposed SEBwill learn the semantic meaning from the task. For example, positive words (good, great, and funny) have positive and high-value similarities with each other, which is also the same case for all negative words (bad, worse, and boring). However, all the negative-positive pairs have negative similarities, which means the subword embedding of our proposed SEBcan automatically learn the semantic meaning.

B.5 ANALYSIS FOR SPACE COMPLEXITY

We analyze the space complexity in the experiments. We mainly take the translation on IWSLT14 and sentiment analysis as examples. Transformer model for translation on IWSLT14 de-en with 256 hidden units in our method SEBas the translation results are close to the traditional subword embedding.

The tables below present the sizes of both the entire model's parameters and the embedding layer for translation on IWSLT and sentiment analysis, respectively. The numbers in () represent the percentage reduction achieved by our method compared to the subword model.

# Params	Whole model	Embedding	BLEU
Subword	37M	5.2M	34.54 ± 0.10
SEB_{co}	33M ($\downarrow$ 12%)	0.7M ($\downarrow$ 94%)	34.62 ± 0.12

Table 8: Transformer model for translation on IWSLT14 de-en.

# Params	Whole model	Embedding	Accuracy
Subword	5.9M	2.4M	81.2 ± 0.7
SEB_{co}	3.8M ($\downarrow$ 36%)	0.8M ($\downarrow$ 68%)	82.5 ± 0.7

Table 9: Sentiment analysis on SST2

# Params	Whole model	Embedding	BLEU
Subword	1.5M	1.5M	85.6 ± 0.5
SEB_{co}	0.4M ($\downarrow$ 72%)	0.5M ($\downarrow$ 80%)	85.8 ± 0.2

Table 10: Sentiment analysis on IMDb

In all of these tasks, our method SEB_{co} can decrease the space complexity, which shows the ability of our method to reduce the model size. In scenarios where model training is necessitated on a device with limited memory, it will be better to make the model smaller while keeping the model's performance.