24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

A Shared-Word Sensitive Sequence-to-Sequence Features Extractor for Sentences Matching

Jinqing Li[1,2] , Dakui Wang[2*] , Xiaojun Chen[2], Pencheng Liao[1,2], Shujuan Chen[3]

Abstract. Sentences matching is a basic task in Natural Language Processing (NLP). Interaction-based methods, which employ interactions between words of two sentences and construct word-level matching features to classify, are generally used due to their fine-grained features. However, they have many invalid interactions that may affect matching precision. In this paper, we limit the objects of interacting to shared words[4] of two sentences. On the one hand, they can reduce invalid interactions. On the other hand, because of the different context semantics, the representation of the same word may be quite different, conversely, the representation difference can also be used to reflect the semantic difference of different contexts. To better extract global features of shared words, we introduce a sequence-to-sequence features extractor to force decoder to learn more contextual information from encoder. We implement the method based on Transformer[28], with syntactic parsing as additional knowledge. Our proposed method achieved better performance than strong baselines and the experiment results also demonstrate the efficiency of sequence-to-sequence features extractor and significance of the shared words.

1 INTRODUCTION

Sentences matching is a basic task in Natural Language Processing(NLP) which is included in many benchmarks, such as GLUE in [29], SNLI in [2], aNLI in [9] and so on. Many of the previous works have provided advanced solutions to the task. Two primary ideas to perform the task are sentence-level matching and word- or phrase-level matching. The former, called presentation-based matching, extracts features of sentences as vectors with diverse neural networks, and then calculates their similarity or feed them into a classifier like [18, 17, 8]. The latter, which is also called interaction-based matching, introduces more detailed matching information about words and enhances the matching ability. [19] is a classical method that proposed a matching pyramid by constructing a similarity matrix between words of texts. With the introductions of attention mechanisms [1] and [15] in NLP, various attention mechanisms and their variants have been widely applied to texts interactive matching, such as [20, 30, 3, 10, 35, 23]. The interaction-based matching methods can obtain more detailed information by comparing words or phrases within two sentences at a finer level, but the method may also be affected by the data distributions so that leads to limitation on generalization ability. The above-mentioned methods, both presentation-based and interaction-based, have one thing in common, that is, an encoder is used to extract the features of words, followed by interacting between words or calculating the representations of sentences, and then performs the corresponding matching calculation.

Large-scale pre-training models have become a trend in recent years, the performance of sentences matching got an unprecedented improvement as well. BERT [4] sets new records on eleven Natural Language Understanding (NLU) tasks which include several sentences matching task. Other variants of BERT also show great improvements, like Roberta [14], XLNet [33], etc. In MTDNN [13], Multi-Task Learning(MTL) also injects vitality into NLU tasks. BART [11] is pre-trained for Neural Language Generation(NLG), translation and comprehension, using denoising sequence-to-sequence model, but achieves comparable results on many NLU tasks as Roberta, XLNet. MASS [24] and T5 [22] also employ a sequence-to-sequence framework to perform pre-training but with different masking strategies and pre-training methods. All the above encoder-decoder based pre-training methods exploit decoder to get more advanced information.

In many NLP tasks, we consider that the same word represents different semantics in different contexts, especially when pre-train word embeddings. For a negative example, a pair sentences *Rose is a beautiful girl* and *Rose is a kind of beautiful flower*, the word *Rose*, one for a girl's name and the other for flower, has different meanings depending on its context. In other words, backing to our matching task, if we know the first *Rose* is saying a girl and the second is saying a flower, we argue that the two sentences are not of the same semantic type. For positive one, two sentences of *How do you get better grades?* and *How can I dramatically improve my grades?*, to the shared words *grades*, the first sentence concerns *get better* and the second concerns *improve* while the two sub-sequences are of same semantic so we argue that they are synonymous. Because of the different context semantics, the representation of the same word may be quite different. Conversely, the representation difference can be used to reflect the semantic difference of the contexts.

Based on the above observations, we propose a new sequence-to-sequence method to extract the contextual features of the shared words of paired sentences, see in Figure 1. We take the **not** shared words and special tokens of $([CLS], [SEP])$ as the input of Encoder and the rested as the input of Decoder. We get the hidden features of shared words from Decoder and match the same word one-to-one correspondence between each of the hidden feature matrices. We then input the obtained feature matching matrix into the classifier after max- or avg-pooling to get the matching type. Our main

[1] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.

[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. Emails:{lijinqing, wangdakui, chenxiaojun, liaopengcheng }@iie.ac.cn. * corresponding author.

[3] China Cybersecurity Review Technology and Certification Center. Email: chensj@isccc.gov.cn.

[4] the same words between two sentences

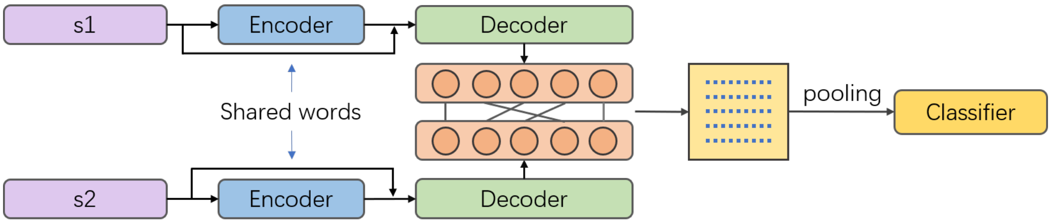


Figure 1: The general architecture of our proposed method (blue arrows are notes).

contributions are as follows:

- To our best knowledge, we are the first to use the shared words contextual differences as matching features of paired sentences for matching task.
- We use encoder and decoder both as features extractor of one sentence.
- Explicit syntactic information is introduced into the Encoder to guide structure features extracting. The experiment results show that it helps a lot with the best performance of our proposed method.

2 RELATED WORK

In this section, we introduce some related previous works and describe them simply.

2.1 Semantic matching

As a fundamental and general NLP task, sentence matching has made rapid progress. Generally, the previous works use encoder as features extractor which is almost based on RNN and its variants. However, the development of attention mechanisms injects great vitality into sentences matching tasks whatever representation-based or interaction-based methods. [25] proposes an iterative refinement LSTM-based encoder for sentence embeddings which is different from traditional stacked-LSTM encoders. [27] introduces a compact and powerful method in which paired sentences are compared, compressed and propagated to enhance the representation learning ability. [6] improves the performance of both long and short sentence embeddings through introducing words distance mask that captures the local dependency which is also implemented based on Transformer. Distance mask is explicitly added to multi-head self-attention weights to learn more about phrase structures. Our proposed method also inspired by it. [12] performs multi-turn inferences of multiple matching features. For each turn, a particular feature instead of all features is focused so that the model can capture the matching information adequately. It demonstrates that fused features will lose much information. It's also why our method works. [26] proposes multiway attention networks in which query information is added into the answer through different forms of attention and they finally use the representation of the answer fused the query as feature of the classifier. [21] significantly improves upon the state of the art in 9 out of the 12 tasks studied by using the 12-layer decoder of Transformer for pre-training and then fine-tuning the model on specific tasks.

2.2 Transformer

Transformer achieved the state of the art results on the WMT 2014 English-to-German translation task and English-to-French translation task when it is proposed. Different from traditional sequence-to-sequence models, the Transformer is a simple and parallelized network using only attention mechanism and models sequence information by position embeddings instead of most used RNN or its variants. As a sequence-to-sequence method, Transformer consists of encoder and decoder including six layers respectively.

For the encoder of Transformer, each layer contains two sublayers: multi-head attention and position-wise feed-forward networks. Multi-head attention consists of several parallelized self attention layers to which scaled dot-product attention weights are calculated as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

where $Q \in R^{n \times d_k}, K \in R^{m \times d_k}, V \in R^{m \times d_v}$, d_v, d_k are dimentions of V and K respectively, n, m are the first dimension size of Q and K, V. Note that $\frac{1}{\sqrt{d_k}}$ is the scaling factor where it differs from dot-product attention. Multi-head attention then concats the output of each head:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2}$$

where $head_j = Attention(QW_j^Q, KW_j^K, VW_j^V)$, $W_j^Q \in R^{d_m \times d_k}, W_j^K \in R^{d_m \times d_k}, W_j^V \in R^{d_m \times d_v}$, d_m is the model's hidden dimension, h is the number of attention heads, and $Q = K = V$ in self-attention. Multi-head attention encourages the model to jointly attend to information from different representation subspaces at different positions. Following the multi-head attention, a fully connected feed-forward networks layer which is applied to each position separately and identically is implemented for linear transformation with ReLU activation:

$$FFN(x) = max\{0, xW_1 + b_1\}W_2 + b_2 \tag{3}$$

Residual connection and layer normalization are used in each sublayer.

The decoder of the Transformer also is composed of six stacked layers as the encoder. A multi-head cross-attention between the outputs of encoder (K, V,in Formula(1)) and multi-head self-attention sublayer of decoder (Q, in Formula(1)) is added to each of decoder layers except the two sublayers mentioned in encoder.

2.3 Syntactic parsing

Syntactic parsing is one of the critical technologies in NLP. It is used to determine the syntactic structures of sentences or the dependency relationships between words of sentences. [3] shows that syntactic parsing information can achieve additional improvement on carefully designed chain LSTMs even other strong models. Another recent research [7] performs a series of experiments to explore the essence

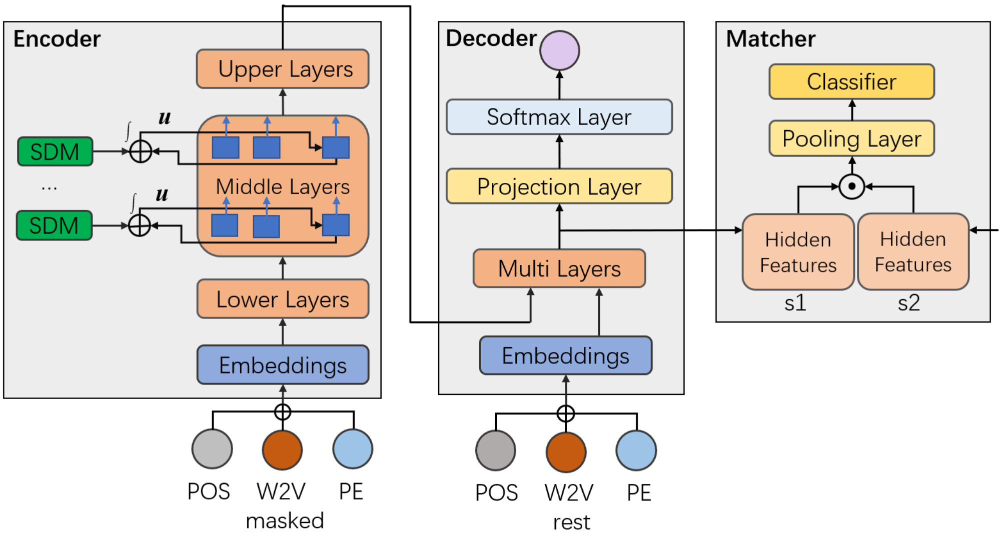


Figure 2: The three main components of our proposed method: Encoder (left), Decoder (middle) and Matcher (right). $\boldsymbol{u}$ is **update**.

of the representations learned by different layers of BERT. The results show that BERT can capture rich linguistic information such as phrase information in lower layers, syntactic information in the middle layers and semantic information in the upper layers. The linguistic information is extremely significant for the strong performance of BERT on diverse language understanding benchmarks. Therefore, one intelligent agent could be more powerful if its features extractor can capture rich and multi-aspect information. Unlike BERT, Transformer has only six stacked layers that may not enough for more plentiful features. Without adding model parameters, it is a good idea to add some auxiliary knowledge. Another reason we leverage syntactic information is that we mask the shared words in the input of encoder which leads to the loosely coupled encoder.

3 METHOD

In this section, we will define the task format, some notations used later and make a detailed description of our proposed method.

3.1 Problem Statement

Given a pair of sentences, s_1 and s_2, the target of this task is to get the matching type $m(s_1, s_2)$, where $m(.) \in \{0, 1\}$, 1 for matching and 0 for not. $s_1 = (w_i)$ for $i \in [0, 1, 2, ..., l_1]$ and $s_2 = (w_j)$ for $j \in [0, 1, 2, ..., l_2]$, where l_1 and l_2 are the lengths of s_1 and s_2 respectively. The shared words are the words existing in both s_1 and s_2 at the same time.

3.2 The proposed model

The proposed method mainly includes three components: Encoder, Decoder and Matcher. Encoder is responsible for extracting rich context information about **not** shared words. The decoder is mainly used to explore representations of the shared words by forcing Encoder to contribute more information about their context. By calculating the representation differences or correlations between paired same words from two sentences, the Matcher obtains the final matching feature and put it into a classifier to get matching type.

3.2.1 Input

As shown in Figure 3, the input embeddings include three parts: word embeddings (W2V), Part-of-Speech tagging embeddings (POS) and position embeddings (PE). POS features are used to help phrases learning in lower layers.

For the Encoder, as shown in the left of Figure 2, the proposed method replaces the shared words with $[blank]$ tokens firstly but still keep their position and POS embeddings. So that we have the complete position information of sentence to perform the dependency guides in the middle layers.

For the Decoder, on the contrary, we only use the shared words as input but the position information is the absolute position in the original sentence. So the length of Decoder input is relatively shorter than that of Encoder.

In the actual experiment, a pair of sentences may exist too many shared words or a few even none except the begin and end tokens of ($[CLS]$, $[SEP]$). So we present three strategies for choosing shared words, with decreasing priority:

- Tokens of $[CLS]$ and $[SEP]$ will always be selected for the semantic comparison of paired sentences. Therefore the input of Decoder contains at least two shared words.
- Tokens **not** in stopwords will be chosen preferentially because the stopwords generally are not so meaningful but modifying.
- Tokens in stopwords will be added to shared words set if there are not enough non-stopwords.

Note that if there are too many shared words, we randomly choose the fixed number of them.

We define W2V as $\boldsymbol{v}$, PE as $\boldsymbol{pe}$ and POS as $\boldsymbol{pos}$, so the input $\boldsymbol{e}$ of Encoder or Decoder can be formulated as:

$$\boldsymbol{e} = Dropout(Layernorm(\boldsymbol{v} + \boldsymbol{pe} + \boldsymbol{pos})) \tag{4}$$

where $Layernorm(.)$ performs layer normalization and $Dropout(.)$ randomly inactive some values.

3.2.2 Encoder

Our proposed method is based on Transformer, but we did some transformations inspired by [7] which demonstrates that the low lay-

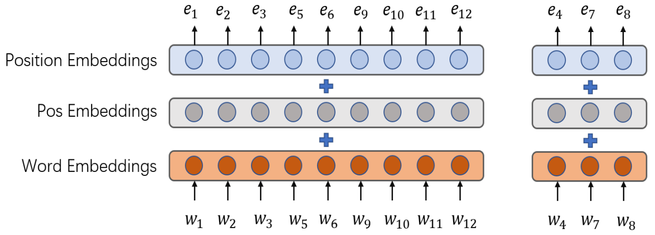

e_1 e_2 e_3 e_5 e_6 e_9 e_{10} e_{11} e_{12}
e_4 e_7 e_8
Position Embeddings
Pos Embeddings
Word Embeddings
w_1 w_2 w_3 w_5 w_6 w_9 w_{10} w_{11} w_{12}
w_4 w_7 w_8

Figure 3: The input of Encoder and Decoder.

ers of BERT will learn more phrases information, the middle ones for syntactic dependency information and the upper ones for semantic information. Now therefore we leverage syntactic parsing obtained from tool *Stanford CoreNLP*[1] to guide the Encoder. We divide the Transformer encoder of six layers into three abstract layers: lower layers, middle layers and upper layers.

For lower layers, we add the POS embeddings into the token embeddings as syntactic guides. The lower layers of Encoder can be formulated as:

$$l_k = MultiHead(f^{en}_{k-1}, f^{en}_{k-1}, f^{en}_{k-1}) \tag{5}$$

$$f^{fc}_k = FFN(Layernorm(l_k + f^{en}_{k-1})) \tag{6}$$

$$f^{en}_k = Layernorm(f^{fc}_k + Layernorm(l_k + f^{en}_{k-1})) \tag{7}$$

where $k \in [1, ..., lower\_index]$ is the integer index of lower layers while $lower\_index$ is the max index, l_k defines the output of multi-head attention, $l_0 = \boldsymbol{e}$ and operation $(+)$ represents residual connection.

The attention mechanism can capture the stucture of text. For middle layers, we add firstly the syntactic dependency matrix, which defines the syntactic structure of the sentence, to some of the attention weights of middle layers multi-head attention. Then we re-softmax the attention weights to get new weights for self-attention representations. Note that we define the syntactic dependency matrix $SDM \in R^{max\_len \times max\_len}$, where max_len is the max length of the encoder input, $SDM(i, j) = 1$ if word w_i depends on w_j otherwise $SDM(i, j) = 0$. We introduce syntactic dependency matrix to guid middle layers for two reasons: First, we replace shared words with $[blank]$ tokens so word embeddings information of them is lost but we still want to get their dependency structure. For the second, syntactic structure is considered significant as one of multi-aspect information. So the middle layers are formulated as:

$$head^k_h = softmax(\frac{Q_h K^T_h}{\sqrt{d_k}} + SDM)V_h \tag{8}$$

$$m_k = MultiHead(f^{en}_{k-1}, f^{en}_{k-1}, f^{en}_{k-1}) \tag{9}$$

$$f^{fc}_k = FFN(Layernorm(m_k + f^{en}_{k-1})) \tag{10}$$

$$f^{en}_k = Layernorm(f^{fc}_k + Layernorm(l_k + f^{en}_{k-1})) \tag{11}$$

where $k \in [lower\_indexlower\_index + 1, ..., middle\_index]$ is the integer index of middle layers while $middle\_index$ is the max index, m_k defines the output of multi-head attention, $Q_h = K_h = V_h$ are the input of the h_th head and the input of first middle layer is the last output of lower layers. We simply add the syntactic dependency matrix to the last head of every middle layer in Formula(8). The upper layers formulations are as same as the lower layers.

3.2.3 Decoder

Original decoder employs two mask strategies: padding mask and sequence mask. The former avoids calculating padding tokens when doing self- or cross-attention while the latter is used to do decoding by not letting the word see the words after itself. In our case, we argue that the Decoder tokens should focus on all words before and after itself for comprehensive information. So like BERT, we employ a bidirectional decoder with shared words of each sentence as inputs. The Encoder and Decoder share word embeddings. Another dissimilitude is that we use less layers to decode, see next section for detail. The Decoder layers are formulated as follows:

$$f^{de}_0 = \boldsymbol{e_{sw}} \tag{12}$$

$$l^s_k = MultiHead(f^{de}_{k-1}, f^{de}_{k-1}, f^{de}_{k-1}) \tag{13}$$

$$ln^s_k = Layernorm(l^s_k + f^{de}_{k-1}) \tag{14}$$

$$l^c_k = MultiHead(f^{en}_6, f^{en}_6, ln^s_k) \tag{15}$$

$$ln^c_k = Layernorm(l^c_k + ln^s_k) \tag{16}$$

$$f^c_k = FFN(ln^c_k) \tag{17}$$

$$f^{de}_k = Layernorm(f^c_k + ln^c_k) \tag{18}$$

where $k \in [1, 2, ..., de\_layers]$, $\boldsymbol{e_{sw}}$ is the embeddings of shared words, l^s_k represents output of multi-head self-attention, l^c_k is multi-head cross-attention output between the output of Encoder and the output of multi-head self-attention in current layer, f^c_k is output of fully connected layer, ln^s_k, ln^c_k and f^{de}_k are residual connection and layer normalization after l^s_k, l^c_k and f^c_k respectively.

3.2.4 Matcher

The primary idea of the proposed method is that we judge whether two sentences are semantically similar by exploring whether the same words in two sentences focus on the similar contents. We implement a comparison in the Matcher component.

As shown in Figure 1, taking the output of Decoder multi-layers as shared words hidden features, we calculate hidden features of same word in two sentences one-to-one correspondence by operation of subtracting or element-wise product to get the matching matrix. For example, we apply element-wise product to hidden decoder states of shared word *helpful* from two sentences and get a result vector of this shared word. Then we pool the matching matrix into a matching vector by max- or average-pooling, and feed it into a linear classifier to get the match type. Note that, the matching matrix is heterogeneous because the matching vector contains both sentences semantic differences and shared words contextual differences. We formulate the Matcher component as follows:

$$M_1 = f^{de,1}_{de\_layers} \tag{19}$$

$$M_2 = f^{de,2}_{de\_layers} \tag{20}$$

$$M = (M^i_1 \circ M^j_2) \quad (where \quad sw^i_1 = sw^j_2) \tag{21}$$

$$\boldsymbol{mv} = pooler(M) \tag{22}$$

$$class = Softmax(FFN(\boldsymbol{mv})) \tag{23}$$

where M_1 and M_2 are the output of Decoder for paired sentences, M is the matching matrix, $\circ$ represents element-wise product, sw^i_1 and sw^j_2 are shared words from s_1 and s_2 respectively where the orders may be different, $\boldsymbol{mv}$ is matching vector obtained by $pooler(.)$ $(max - pooling$ or $avg - pooling)$ and $class$ is the matching type probability distribution of the input paired sentences.

[1] https://stanfordnlp.github.io/CoreNLP/

4 EXPERIMENTS

We implement our experiments based on Transformer on three classical datasets: QQP[1,2], SciTail[3] and SNLI[4]. And our proposed method performs better than some strong baselines. The following are detailed introductions of our experimental datasets, settings, optimization object, results and ablation experiments.

4.1 Datasets

4.1.1 QQP

QQP (Quora Question Pairs) dataset contains over 400,000 questions duplicate pairs from the community question-answering website Quora (train: 364k, test: 391k). The task belongs to paraphrase tasks, which is used to judge whether the paired questions are equivalent in semantic meaning. However, the dataset is unbalanced with about 63% negative examples and 37% positive examples. Each pair of questions is assigned a label 1 for semantic equivalence and 0 for not. [29] offers detailed information of it and gives a baseline of it which gets the best performance by the experiment of $LSTM + Attn + Elmo$ with a single task.

4.1.2 SciTail

In [9], SciTail dataset is first introduced. It is the first entailment dataset that is acquired from science question answering created by converting a multiple-choice question and the correct answer choice into an assertive statement to form the hypothesis. 27k pairs of sentences are included in it (train: 23.6k, dev: 1.3k, test: 2.1k). SciTail is considered a textual entailment task in which it contains two target labels: entails and neutral. [9] provides a baseline of it which is called by $DGEM$ (Decomposed Graph Entailment Model).

4.1.3 SNLI

SNLI (the Stanford Natural Language Inference) is a manually written corpus which consists of 570k pairs of English sentences (train: 550k, dev: 10k, test: 10k). It contains three label types: entailment, contradictory and neutral. Supporting natural language inference, also known as recognizing textual entailment, the task corresponding to the dataset is to determine whether the given paired sentences are semantic entailment or contradictory or neutral. [2] introduces the corpus in detail and gives a baseline first time. Note that, different from the above two, the task is a three-way classification.

4.2 Loss optimization object

The sequence-to-sequence framework is generally employed in NLG. Therefore, maximizing the log probability of correctly decoding shared words given the rest words of the input sentence is our first goal. We define Decoder loss as:

$$L_{decoder} = -\sum_{sw^i \in sw} \log P(sw^i | sw^-) \tag{24}$$

where sw^- is **not** shared words. Our final goal is judging the semantic matching type by a classifier. So the other loss function is given by a cross-entropy:

$$L_{classify} = -\sum\nolimits_{i=0}^{class\_num-1} y_i \log(y_i^-) \tag{25}$$

where $class\_num$ is the number of matching types, y is the ground truth and $y^- = class$ is the prediction result. The overall optimal object is:

$$L = \alpha L_{decoder} + \beta L_{classify} \tag{26}$$

where α and β are weight hyperparameters.

4.3 Experiment settings

In our experiments, we employ the Transformer as our base code and apply a series transformations on it.

We define our Encoder of 6 layers of multi-head attention and feed-forward network that is the same as the original Transformer except we add a syntactic dependency matrix to its middle layers. Of each layer of Encoder and Decoder, we use 8 heads, $512d$ of W2V, PE and POS embeddings, $2048d$ for the first layer of FFN sublayer and $512d$ for the last7 layer of FFN sublayer, $d_k = 64$. We use dropout $dr = 0.1$, $lr = 1e-5$ for learning rate. For the loss weights of two types, we use $\alpha = 0.04$ and $\beta = 0.96$ empirically. We use Adam optimizer with initial learning rate lr and reduce it dynamically along with the training process with decay factor $decay\_factor = 0.8$ if there is no improvement on accuracy, f1 or loss value for $patience$[5] iterations. For the datasets of QQP and SNLI, $class\_num = 2$ but $class\_num = 3$ for SciTail. We use $max - pooling$ for pooling operation and element-wise product for features interaction because we found them perform better in experiments.

Also, we'll explore the dividing position of the Encoder layers for best performance from which the Encoder is divided into lower, middle and upper abstract layers. As for the number of Decoder layers and the shared words sequence length, we'll study them respectively as well.

4.4 Results

In this section, we will report our experimental results on several explorations including whether shared words matter or not, how long the sequence of the shared words should be, how to divide the Encoder into 3 abstract layers and how many layers for the Decoder. We also show some strong baseline results and our ablation experiment results.

4.4.1 Significance exploration of shared words

We experiment with an encoder to verify the significance of shared words. We design three models to perform it. For the first model, we apply Transformer encoder as feature extractor and calculate the element-wise product of each word in s_1 with all words in s_2 followed by $max - pooling$ which leads to one word matching vector. After acquiring all words matching vectors, we employ $max - pooling$ on them so that we obtain the one of the matching vectors. For the sake of symmetry, we do the same operations by

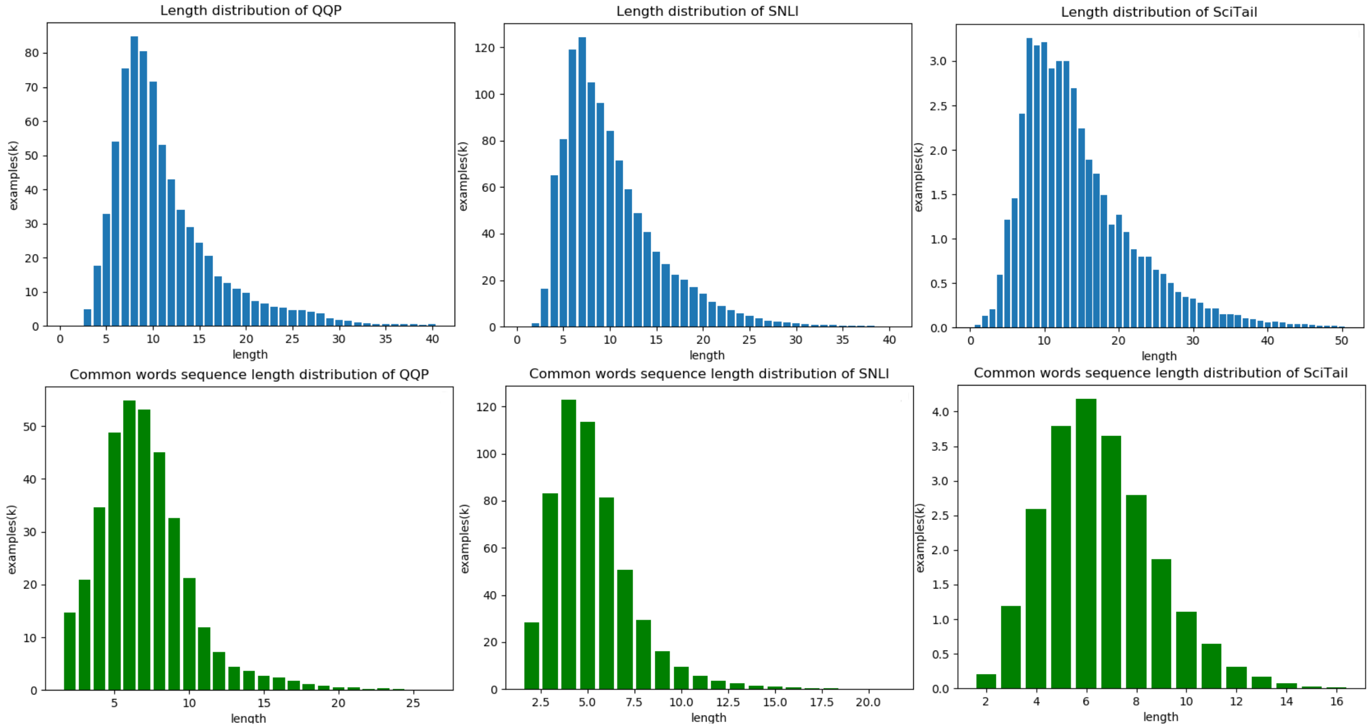
[1] https://gluebenchmark.com/

[2] https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs

[3] https://leaderboard.allenai.org/scitail/submissions/public

[4] https://nlp.stanford.edu/projects/snli/

[5] changes with dataset, batch_size, evaluate steps



Length distribution of QQP

Length distribution of SNLI

Length distribution of SciTail

Common words sequence length distribution of QQP

Common words sequence length distribution of SNLI

Common words sequence length distribution of SciTail

Figure 4: The sentence length distributions (top) and shared words sequence length distributions (bottom) for paired sentences ($[CLS]$ and $[SEP]$ included) on three datasets (left: QQP, middle: SNLI and right: SciTail).

Table 1: The accuracy results on dev sets of three models: Tr_En_ALL (Transformer Encoder and ALL words participate in interactions), Tr_En_SWs (Transformer Encoder and only Shared Words participate in interactions) and Ours (our proposed method)

model	QQP	SNLI	SciTail
Tr_En_ALL	85.1	85.1	86.5
Tr_En_SWs	85.3	85.5	86.9
Ours	**88.5**	**88.4**	**87.7**

swapping two sentences. Then we concat the two matching vector as final feature and put it into classifier.

The above model interacts all words between the s_1 and s_2. To explore whether shared words really matter, we set up the second comparison experiment. We just interact with the shared words one-to-one correspondence for the paired sentences. The shared words are chosen by the strategies in section 3.1.1. We calculate the element-wise product of a pair of same words from different sentences to get a matching matrix. Then we apply max-pooling on matching matrix and feed the resulting vector into a linear classifier.

To compare with the models mentioned above, settings of two layers of Decoder, 20% or 30% length of shared words and $[2, 4]$ dividing indexes of Encoder are applied to our proposed method in this section. The results in Table 1 show that the method interacting between shared words performs better than that between all words. More noises are added into the final matching feature so that the information which really matters is diluted. However, our proposed method achieves the best performance of the three methods. This further validates that using the only encoder to extract information of a word is not enough. Generating shared words in the decoder can actually force the decoder to learn more helpful contextual information from the encoder.

4.4.2 The max length of Decoder input

In our task, the sentences are usually short. In Figure 4 top we exhibit the sentence length distributions and the bottom is the distributions of shared words sequence length for the paired sentences on three datasets. In this case, how long the Decoder input should be set is of prime importance because hundreds or thousands of original sentences contain less than 5 words. There are three extremes: the Encoder input is too short, the second one, the Decoder input is too short which only consists of $([CLS], [SEP])$ and the last one for both. Here, if the Encoder input is too short, we don't mask it simply and if the target is too short, the decoder degenerates to contain only semantic vectors. For the general case, we need to develop a reasonable strategy to define the shared words sequence length. Obviously, the tendency towards the two distributions (Figure 4 top and bottom respectively) are consistent. This means that we need to choose the length of the sequence of the shared words proportional to the sentence length, rather than all the paired sentences share the same length.

For two sentences, we define L is the min length of them ($[CLS]$, $[SEP]$ included). The max length of shared words sequence at most is defined as:

$$L_{sw} = max\{min\{L \times factor, max\_length\}, 2\} \tag{27}$$

where $factor$ is in $[0.1, 0.2, 0.3, 0.4]$, max_length is a hyper parameter of max Decoder input length and $L\_sw \geq 2$. In this section, we set 2 layers of Decoder and $[2, 4]$ diving indexes of Encoder. Table 2 shows the results with different $factor$. Longer input of Decoder can cause performance degradation, which illustrates that if the $factor$ is too big, more stopwords will be included while they may not have enough contextual differences. The results also suggest that masking too many words of Encoder input sequence is not conducive to contextual information extraction.

Table 2: The accuracy results on dev sets for exploration of $factor$ (left) and the dividing way of Encoder layers (right) where (1,234,56) represents the lower layer(s) contain(s) the 1st layer,the middle layer(s) contain(s) the 2nd,3rd,4th layers and the upper layer(s) contain(s) the 5th,6th layers.

$factor$	QQP	SNLI	SciTail	$dividing$	QQP
0.1	85.9	86.4	79.2	(12,34,56)	**88.5**
0.2	**88.5**	88.1	**87.7**	(1,234,56)	87.3
0.3	87.3	**88.4**	87.3	(12,345,6)	87.7
0.4	87.2	87.6	83.9	(1,2345,6)	88.0

4.4.3 The dividing positions of Encoder layers

We divide the Encoder layers into 3 abstract layers: the lower,middle and upper layers. For contrast, we employ four dividing strategies to test their performance. As shown in the right of Table 2, we acquire the best dividing positions of $(12, 34, 56)$(see also $[2, 4]$). This results partly tell us that phrase information, syntactic information and semantic information are almost equally significant in semantic matching tasks therefore we need comparative number of parameters to fit them.

4.4.4 The number of Decoder layers

In the process of the experiments, we found that different numbers of Decoder layers matter a lot for the experiment results. Therefore, we perform different numbers from 1 to 6 to test their performance. See the results in Table 3, when we use 2 or 3 layers of Decoder, it achieved best performance. We argue that this is because the shared words sequence is too short. The more layers, the more capacity the model owns, but we can't fit it better with a few words.

Table 3: The accuracy results on dev sets for exploration of the number of Decoder layers (de_layers).

de_layers	QQP	SNLI	SciTail
1	87.3	86.7	83.3
2	**88.5**	88.1	**87.7**
3	88.1	**88.4**	86.2
4	84.1	83.5	82.7
5	79.7	78.6	80.5
6	78.9	77.4	79.7

4.4.5 Best performance on three datasets

We test our method of optimal configuration on the above datasets respectively. We also list some results of previous works' results as our strong baselines of which attention-based methods are applied in Table 4. In the listed previous works, [3, 9, 27, 12, 29, 26, 5, 32] employ interaction-based methods like us. Without losing generality, we also list some representation-based methods as [21, 6]. The results demonstrate that our proposed method is effective and robust. On the other hand, it also shows the strong features extraction ability of the Transformer. Note that FTLM[21] is Finetuned Transformer Language Model which improves natural language understanding by generative pre-training. In their pre-training task, they use a 12 layers Transformer decoder as a language model which also inspired us for our proposed method. However, as the last listed line in Table 4, our results are still quite far from the best-reported models StructBERT [31], Semantics-aware BERT [34] and MTDNN [13], which used large-scale pre-trained models or multi-task models based on BERT [4]. This is also a performance limitation of the traditional methods compared with the large-scale pretrained or multi-task advanced methods.

Table 4: The accuracy results on QQP, SNLI, SciTail test sets. The last line is the best result of each of the datasets until 2020/02/01.

model	QQP	model	SNLI	model	SciTail
Base[29]	86.5	Base[16]	77.3	Base[9]	77.0
BiMPM[32]	88.1	HIM[3]	88.6	CAFE[27]	83.3
DIIN[5]	**89.1**	MIMN[12]	89.3	MIMN	84.0
MwAN[26]	**89.1**	MwAN	89.4	FTLM[21]	88.3
Ours	**89.1**	Ours	**89.7**	Ours	**88.9**
Best[31]	91.0*	Best[34]	91.9*	Best[13]	96.1*

4.5 Ablation experiments

In this section, we further analyze whether the syntactic information really matters or not. We perform three experiments: removing the POS embeddings from inputs of Encoder and Decoder, removing the syntactic dependency matrix from the middle layers of Encoder and removing both. The resultss in Table 5 demonstrate that syntactic dependency matrix is critical for the masking words of Encoder input. The POS embeddings make a little contribution to our best performance.

Table 5: The accuracy results on QQP dev set of ablation experiments. (-) represents remove corresponding item(s) from the whole model.

Dataset	Ours	-POS	-SDM	-POS-SDM
QQP	**88.5**	87.8	85.7	85.1

5 CONCLUSION

In this work, we propose a share-word sensitive sequence-to-sequence features extractor based on Transformer which includes three main components: Encoder, Decoder and Matcher. Different from the traditional sentence matching method, we employ the Decoder to learn more contextual information of shared words from Encoder. Because we replace the shared words embeddings of Encoder with $[blank]$ embeddings, it's more difficult to learn structure information. Therefore we also introduce syntactic analysis information to the Encoder as guides. We divide the Encoder into three abstract layers: the lower, middle and upper layers. The lower layers are for basic phrase information, the middle layers are for the dependency information while the upper layers are for the semantic information. Evaluations on the significance of shared words demonstrate that shared words in paired sentences matter a lot for the semantic matching. The length of the sequence of the shared words is a crucial factor when we actually practice it. The exploration of the number of Decoder layers reminds us of the matching of networks' capacity and data size is important. Our sequence-to-sequence features extractor develops a new idea for sentence matching or other NLU tasks while it remains yet to be developed. Compared with some other classical methods, our method has achieved better results. However, a mass of works should be explored for sequence-to-sequence feature extraction which is also the next research plan of us, such as how to combine it with large-scale pre-training and how to jointly learn with other tasks like multi-task learning.

ACKNOWLEDGEMENTS

Supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No.XDC02040400

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, 'Neural machine translation by jointly learning to align and translate', *CoRR*, **abs/1409.0473**, (2014).

[2] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning, 'A large annotated corpus for learning natural language inference', in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, (2015).

[3] Qian Chen, Xiao-Dan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen, 'Enhanced lstm for natural language inference', in *ACL*, (2016).

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, 'Bert: Pre-training of deep bidirectional transformers for language understanding', in *NAACL-HLT*, (2019).

[5] Yichen Gong, Heng Luo, and Jian Zhang, 'Natural language inference over interaction space', *ArXiv*, **abs/1709.04348**, (2017).

[6] Jinbae Im and Sungzoon Cho, 'Distance-based self-attention network for natural language inference', *ArXiv*, **abs/1712.02047**, (2017).

[7] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah, 'What does bert learn about the structure of language?', in *ACL*, (2019).

[8] Tom Kenter, Alexey Borisov, and Maarten de Rijke, 'Siamese cbow: Optimizing word embeddings for sentence representations', *ArXiv*, **abs/1606.04640**, (2016).

[9] Tushar Khot, Ashish Sabharwal, and Peter Clark, 'Scitail: A textual entailment dataset from science question answering', in *AAAI*, (2018).

[10] Wuwei Lan and Wei Xu, 'Neural network models for paraphrase identification, semantic textual similarity, natural language inference, and question answering', in *COLING*, (2018).

[11] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke S. Zettlemoyer, 'Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension', *ArXiv*, **abs/1910.13461**, (2019).

[12] Chunhua Liu, Shan Jiang, Hainan Yu, and Dong Yu, 'Multi-turn inference matching network for natural language inference', in *CCF International Conference on Natural Language Processing and Chinese Computing*, pp. 131–143. Springer, (2018).

[13] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao, 'Multi-task deep neural networks for natural language understanding', in *ACL*, (2019).

[14] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov, 'Roberta: A robustly optimized bert pretraining approach', *ArXiv*, **abs/1907.11692**, (2019).

[15] Thang Luong, Hieu Pham, and Christopher D. Manning, 'Effective approaches to attention-based neural machine translation', in *EMNLP*, (2015).

[16] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal, 'Can a suit of armor conduct electricity? a new dataset for open book question answering', in *EMNLP*, (2018).

[17] Jonas Mueller and Aditya Thyagarajan, 'Siamese recurrent architectures for learning sentence similarity', in *AAAI*, (2016).

[18] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru, 'Learning text similarity with siamese recurrent networks', in *Rep4NLP@ACL*, (2016).

[19] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng, 'Text matching as image recognition', in *AAAI*, (2016).

[20] Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit, 'A decomposable attention model for natural language inference', in *EMNLP*, (2016).

[21] Alec Radford, 'Improving language understanding by generative pre-training', (2018).

[22] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu, 'Exploring the limits of transfer learning with a unified text-to-text transformer', *arXiv e-prints*, (2019).

[23] Jinfeng Rao, Linqing Liu, Yi Tay, Wei Yang, Peng Shi, and Jimmy Lin, 'Bridging the gap between relevance matching and semantic matching for short text similarity modeling', in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5373–5384, (2019).

[24] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu, 'Mass: Masked sequence to sequence pre-training for language generation', in *ICML*, (2019).

[25] Aarne Talman, Anssi Yli-Jyrä, and Jörg Tiedemann, 'Sentence embeddings in nli with iterative refinement encoders', *Natural Language Engineering*, **25**, 467–482, (2019).

[26] Chuanqi Tan, Furu Wei, Wenhui Wang, Weifeng Lv, and Ming Zhou, 'Multiway attention networks for modeling sentence pairs', in *IJCAI*, (2018).

[27] Yi Tay, Anh Tuan Luu, and Siu Cheung Hui, 'Compare, compress and propagate: Enhancing neural architectures with alignment factorization for natural language inference', in *EMNLP*, (2017).

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, 'Attention is all you need', in *NIPS*, (2017).

[29] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman, 'GLUE: A multi-task benchmark and analysis platform for natural language understanding', (2019). In the Proceedings of ICLR.

[30] Shuohang Wang and Jing Jiang, 'A compare-aggregate model for matching text sequences', *ArXiv*, **abs/1611.01747**, (2016).

[31] Wei Wang, Bin Bi, Ming Yan, Chen Wu, Zuyi Bao, Liwei Peng, and Luo Si, 'Structbert: Incorporating language structures into pre-training for deep language understanding', *ArXiv*, **abs/1908.04577**, (2019).

[32] Zhiguo Wang, Wael Hamza, and Radu Florian, 'Bilateral multi-perspective matching for natural language sentences', in *IJCAI*, (2017).

[33] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le, 'Xlnet: Generalized autoregressive pretraining for language understanding', *ArXiv*, **abs/1906.08237**, (2019).

[34] Zhuosheng Zhang, Yu-Wei Wu, Zhao Hai, Zuchao Li, Shuailiang Zhang, Xi Zhou, and Xiaodong Zhou, 'Semantics-aware bert for language understanding', *ArXiv*, **abs/1909.02209**, (2019).

[35] Xiangyang Zhou, Lu Li, Daxiang Dong, Yi Liu, Ying Chen, Wayne Xin Zhao, Dianhai Yu, and Hua Wu, 'Multi-turn response selection for chatbots with deep attention matching network', in *ACL*, (2018).