Multi-Level Head-Wise Match and Aggregation in Transformer for Textual Sequence Matching

Shuohang Wang,[1,2*] **Yunshi Lan,**[1] **Yi Tay,**[3†] **Jing Jiang,**[1] **Jingjing Liu**[2]
[1]Singapore Management University, [2]Microsoft Dynamics 365 AI Research, [3]Nanyang Technological University
{shwang.2014, yslan.2015, jingjiang}@smu.edu.sg, ytay017@e.ntu.edu.sg, jingjl@microsoft.com

Abstract

Transformer has been successfully applied to many natural language processing tasks. However, for textual sequence matching, simple matching between the representation of a pair of sequences might bring in unnecessary noise. In this paper, we propose a new approach to sequence pair matching with Transformer, by learning head-wise matching representations on multiple levels. Experiments show that our proposed approach can achieve new state-of-the-art performance on multiple tasks that rely only on pre-computed sequence-vector-representation, such as SNLI, MNLI-match, MNLI-mismatch, QQP, and SQuAD-binary.

Introduction

Textual sequence matching is important for many natural language processing tasks, such as textual entailment (Bowman et al. 2015), paraphrase identification (Dolan and Brockett 2005), question answering (Tan et al. 2015), etc. There has been a large amount of work focusing on this problem, from exploring classic human-crafted features (Wan et al. 2006), to tree-based neural structures (Mou et al. 2016; Tai, Socher, and Manning 2015), and a large number of attention-based models (Rocktäschel et al. 2015; Yin et al. 2015; Parikh et al. 2016; Lin et al. 2017). Table 1 provides two examples of real sequence matching problems on duplicated question detection and text entailment.

Across the rich history of sequence matching research, there have been two main streams of studies for solving this problem[1]. The first utilizes a sequence encoder (e.g., LSTM/CNN) to obtain static low-dimensional vector representations of sequence pairs. Subsequently, a parameterized function (e.g., Multi-layered Perceptron) is applied on top of the representations to learn a matching score (Bowman et al. 2015; Mou et al. 2016; Tai, Socher, and Manning 2015; Lin et al. 2017). The second direction learns to aggregate word and phrase level interactions using cross-sentence attention, learning the entire matching function in an end-to-end fashion (Rocktäschel et al. 2015; Wang and Jiang 2016; Yin et al. 2015; Devlin et al. 2019).

Q1	Is there a reason why we should travel alone?
Q2	What are,some reasons to travel alone?
L	is_duplicated
P	Two young children in blue jerseys, one with the number 9 and one with the number 2 are standing on the wooden steps in a bathroom and washing their hands in a sink.
H	Two kids in numbered jerseys want their hands.
L	Entailment

Table 1: Examples from Quora Question Pairs (QQP) dataset for duplicated question detection and Stanford Natural Language Inference dataset for text entailment. “Q1” and “Q2” are the question pair. “P” represents the premise and “H” the hypothesis. “L” is the label.

While models that utilize cross-sentence attention typically achieve better performance (Devlin et al. 2019), the encoders are unfortunately not re-usable. Hence, new sequence pairs require re-computation of word/phrasal alignments. Consequently, this incurs additional computation costs. Conversely, models that learn static representations enjoy reuse of their fixed vector representations and are therefore inherently more efficient (i.e., we only need to compute the scoring function on top of the pre-computed representations). As pointed out by Reimers and Gurevych (2019), finding the most similar sentence pair in a collection of n = 10,000 sentences need to run original BERT $n \times (n-1)/2 = 49,995,000$ times. By making use of static representations, we only need to run BERT n times together with some much cheaper matrix multiplications, which will reduce the running time from 65 hours to 5 seconds. Moreover, this inherent scalability benefit enables querying of up to millions of documents (Das et al. 2019) or answers (Seo et al. 2018), which provides a greater recall/coverage in open domain question answering. Our work, inspired by this advantage, focuses on improving

[*]Work done when the author was at Singapore Management University

[†]Now working at Google

[1]As classified in the leaderboard of SNLI, https://nlp.stanford.edu/projects/snli/.

techniques based on learning with pre-computed representations.

Notably, there are several well-established methods for matching pre-computed vector representations (e.g., cosine similarity (Tan et al. 2015), element-wise subtraction (Tai, Socher, and Manning 2015) and direct concatenation followed by Multi-layered Perceptron (Bowman et al. 2015)). Although these matching methods have demonstrated reasonable success for LSTM (Tai, Socher, and Manning 2015) or CNN encoders (Mou et al. 2016), they are not ideal for matching representations obtained from Transformer encoders (Vaswani et al. 2017). Different from other types of encoders, the multi-headed self-attention mechanism that lives at the heart of the Transformer model requires special treatment. To this end, how to effectively aggregate information from multiple heads and multiple layers of the Transformer remains an unanswered research question.

This work is mainly concerned with (1) empirically studying the behavior of the multi-head attention in pretrained Transformer models (Devlin et al. 2019) and (2) proposing a new matching aggregation scheme based on our empirical observations. More specifically, based on our empirical analysis and visualization of the behavior of the underlying multi-headed attention, we arrive at several conclusions. Firstly, most of the heads will assign higher attention weights to specific words. Secondly, heads from different layers target at different aspects of the sequence. Thirdly, some heads pay attention to the same or related words. Lastly, some heads pay attention to stopwords or punctuation marks. Intuitively, it seems as though the representation ability of the Transformer encoder is spread out across the network, heads, and layers. Therefore, specialized aggregation is necessary.

As per our observations, different heads may cover different aspects of information. Therefore, mixing the heads together with a fully connected layer may be sub-optimal. Instead, we propose matching the corresponding heads from different sequences independently and then aggregating them for sequence matching. This is in a similar spirit to the compare-aggregate framework with cross-sequence attention (Wang and Jiang 2016; Parikh et al. 2016), albeit largely based on the head level representations.

Our contributions The contributions of our work can be summarized as follows:

- We propose a new head-wise matching method, specifically designed for the Transformer model (Vaswani et al. 2017; Devlin et al. 2019).
- We investigate the utility of different matching functions for head-wise matching. Based on our experiments, we find that element-wise matching representation works the best for head-wise matching.
- We show that we are able to further boost the performance by making use of the head matching in different layers of Transformer.
- We explore different methods to integrate the head-wise matching representation, arriving at the conclusion that max pooling based method works the best.
- We achieve state-of-the-art results on multiple tasks, including SNLI (Bowman et al. 2015), QQP (Wang et al. 2018), MNLI (Williams, Nangia, and Bowman 2018), SQuAD-binary (Rajpurkar, Jia, and Liang 2018), relying on sentence vector representations.

Related Work

Learning the relationship between textual sequence pairs is a fundamental problem in natural language processing (NLP). A wide range of NLP tasks fit into this problem formulation, ranging from textual entailment (Bowman et al. 2015) to question answering (Tan et al. 2015).

The dominant approaches to sequence matching are largely based on cross-sentence attention (Wang and Jiang 2016; Parikh et al. 2016; Rocktäschel et al. 2015). The key idea is to learn word/phrasal alignment between sequence pairs. Models that compute such alignment on the fly can perform well on many benchmarks. However, the inability to reuse representations hampers its usability in real-world applications.

To mitigate this issue, pre-computed vector representation can be used, due to its ability for reuse. Seo et al. (2018) proposed phrase-indexed question answering (PIQA), a special setting for reading comprehension that prohibits cross-sentence attention being computed on the fly. Instead, representations should be pre-computed and "cached". This provides an intrinsic scalability benefit, enabling efficiency during inference. In practice, requiring the re-computation of alignment on the fly is not suitable for many real-world applications. Das et al. (2019) proposed multi-step reasoning using pre-computed vector representations, which can accelerate searching through documents.

Fine-tuning is another option (Devlin et al. 2019). The key idea is to pretrain a Transformer model on a large Web corpus with unsupervised objectives. These large pre-trained models are then fine-tuned for downstream tasks. The state-of-the-art pre-trained models such as BERT (Devlin et al. 2019) and GPT (Radford et al. 2018) all fine-tune over sequence pairs, and apply self-attention across concatenated pairs. Effectively, these models can compute alignment on the fly, but are expensive for real world applications due to the need of computing token-level cross attention in each forward pass. In this paper, we will investigate the usage of these pre-trained models as static encoders.

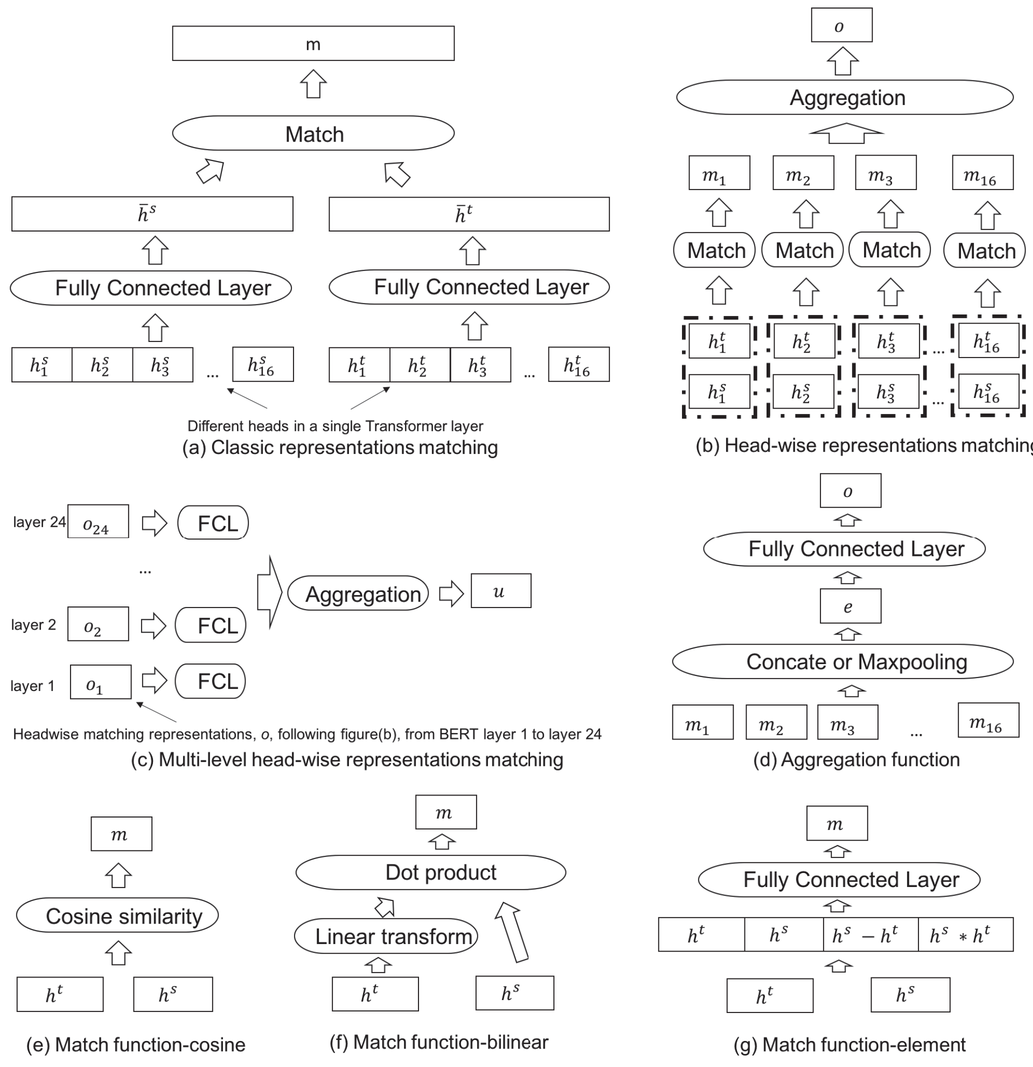
Model

In this section, we will introduce the details of our model. An overview of model architecture is shown in Figure 1.

Transformer Encoder

In this paper, we will focus on making use of the vectorized representations built by Transformer (Vaswani et al. 2017), which is initialized by BERT (Devlin et al. 2019), for sequence matching.

Considering the reusability of encoders, we will not directly concatenate the sequences and run a single Transformer on it for matching. Instead, without cross-sequence attention, we run Transformer on sequence pairs independently and



m
Match
$\bar{h}^s$
$\bar{h}^t$
Fully Connected Layer
Fully Connected Layer
h_1^s h_2^s h_3^s ... h_{16}^s
h_1^t h_2^t h_3^t ... h_{16}^t
Different heads in a single Transformer layer
(a) Classic representations matching
o
Aggregation
m_1 m_2 m_3 m_{16}
Match
Match
Match
Match
h_1^t h_2^t h_3^t ... h_{16}^t
h_1^s h_2^s h_3^s ... h_{16}^s
(b) Head-wise representations matching
layer 24 o_{24} FCL
...
layer 2 o_2 FCL
layer 1 o_1 FCL
Aggregation
u
Headwise matching representations, o, following figure(b), from BERT layer 1 to layer 24
(c) Multi-level head-wise representations matching
o
Fully Connected Layer
e
Concate or Maxpooling
m_1 m_2 m_3 ... m_{16}
(d) Aggregation function
m
Cosine similarity
h^t h^s
(e) Match function-cosine
m
Dot product
Linear transform
h^t h^s
(f) Match function-bilinear
m
Fully Connected Layer
h^t h^s $h^s - h^t$ $h^s * h^t$
h^t h^s
(g) Match function-element

Figure 1: An overview of the model. Fully connected layer (FCL) is a non-linear transformation layer. Figures (e)(f)(g) are different Match functions applied in the first three models.

only match sequences based on their vectorized representations. This setting is useful for large-scale answer extraction (Seo et al. 2018) and multi-step reasoning (Das et al. 2019).

Each Transformer layer is constructed by multi-head self-attention (Lin et al. 2017):

$$\text{MultiHead}(\mathbf{H}) = f\left(\text{Concat}\left(\textbf{head}_1, ..., \textbf{head}_\text{I}\right)\right), \quad (1)$$

where $\mathbf{H}$ is the hidden representation from last layer. $f(.)$ is a non-linear transformation, and the function Concat$(\cdot,\cdot)$ is to concatenate all the head$_i$, which is collected by attention mechanism:

$$\textbf{head}_i = \text{Attention}(\mathbf{H}\mathbf{W}_i^\text{Q}, \mathbf{H}\mathbf{W}_i^\text{K}, \mathbf{H}\mathbf{W}_i^\text{V}), \quad (2)$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}, \quad (3)$$

where $\mathbf{W}_i^\text{Q}, \mathbf{W}_i^\text{K}, \mathbf{W}_i^\text{V}$ are the weights to learn and d is the hidden size of head$_i$ for scaling the attention weights. In the default setting of Transformer, the hidden representation of the first token, which is an inserted special token "[CLS]" for classification, represents the whole sequence as follows:

$$\overline{\mathbf{h}} = \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V})[0], \quad (4)$$

where $[0]$ is to select the vector in the first position. When we need to match a sequence pair, such as in the example in Table 1, the hidden representations of the pair can be obtained as $\overline{\mathbf{h}}^\text{s}$ and $\overline{\mathbf{h}}^\text{t}$, respectively, which can be used as the input of a match function to build the matching representation.

Head-wise Match

For the most widely used encoders, such as CNN (Kim 2014), LSTM (Hochreiter and Schmidhuber 1997), tree-LSTM/CNN (Tai, Socher, and Manning 2015), the representations of different sequences, $\overline{\mathbf{h}}^s$ and $\overline{\mathbf{h}}^t$, will be concatenated and followed by a MLP layer for classification, as shown in Figure 1 (a). While, for the Transformer (Vaswani et al. 2017), it is constructed by the self-attention heads. And each head can focus on a specific aspect of the sequence, as the distribution of self-attention weights are usually quite sharp, as shown in Figure 3. This is the special property for Transformer. In this way, besides directly matching the sequence level representations of Transformer, we can first match the corresponding heads in different sequences and then aggregate the head-wise matching representations to build the sequence level matching representation. An overview of the model is shown in Figure 1 (b).

Specifically, instead of mixing up different heads to get a better sequence level representation, as shown in Eqn.(1), we directly match the $\textbf{head}_i$ from different sequences. For simplicity, we use h_i to represent the first vector, which corresponds to the first token of the sequence in the ith head:

$$\mathbf{h}_i = \textbf{head}_i[0], \tag{5}$$

$\mathbf{h}_i$ represents the i aspect of the sequence information. Instead of merging the heads from one sequence, we first match all heads, $\mathbf{h}_i^{\text{s}}, \mathbf{h}_i^{\text{t}}$, from the sequence pair:

$$\mathbf{m}_i = \text{Match}(\mathbf{h}_i^{\text{s}}, \mathbf{h}_i^{\text{t}}) \tag{6}$$

where $\mathbf{m}_i$ is the head-wise matching representation. Then another layer is used to aggregate the matched heads:

$$\mathbf{o} = \text{Aggregation}\left(\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_{\text{I}}\right), \tag{7}$$

where $\mathbf{o}$ will be used for sequence level matching. The Match and Aggregation functions will be introduced in Eqn.(9-13).

Multi-level Head-wise Match

Different layers of the encoder can represent different levels of information. Both the shallow and deep structures are useful for the sequence matching. Moreover, based on our visualization on the attention weights, we find that the heads in different layers will also pay attention to different aspects of the sequence. In this way, we can get the head-wise match for every layer in the Transformer. We use $\mathbf{o}_l$ to represent the head-wise matching representation by making use of all the heads in the k$^{\text{th}}$ layer, as shown in Figure 1 (c). And we aggregate the representations from different layers as follows:

$$\begin{aligned} \mathbf{v}_i &= \text{ReLU}(\mathbf{o}_i\mathbf{W}^{\text{v}} + \mathbf{b}^{\text{v}}), \\ \mathbf{u} &= \text{Aggregation}\left(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_k\right), \end{aligned} \tag{8}$$

where $\mathbf{W}^v$ and $\mathbf{b}^v$ are the parameters to optimize. And the $\mathbf{u}$ is the final representation of the sequence matching, which consists of multi-level head-wise match, which can be used for the final classification.

Match and Aggregation Functions

To the best our knowledge, this is the first work to aggregate the head-wise matching in Transformer for sequence matching. We have a further explorations on the influence of different head-wise matching functions, in Eqn. (6), and aggregation functions, in Eqn. (7,8) , for building the final sequence matching representation.

For the **match functions** with two vectors, $\mathbf{h}^{\text{s}}, \mathbf{h}^{\text{t}}$, as inputs, the Cosine similarity would the most efficient one without parameters to learn:

$$m = \text{Match}(\mathbf{h}^{\text{s}}, \mathbf{h}^{\text{t}}) = \text{Cosine}(\mathbf{h}^{\text{s}}, \mathbf{h}^{\text{t}}), \tag{9}$$

where the output is a scalar value. Another more complicated matching would a bilinear which is also widely used for the representation matching or attention weight computing:

$$m = \text{Match}(\mathbf{h}^{\text{s}}, \mathbf{h}^{\text{t}}) = (\mathbf{h}^{\text{s}}\mathbf{W}^{\text{b}}) \cdot \mathbf{h}^{\text{t}}, \tag{10}$$

where $\mathbf{W}^{\text{b}}$ is the matrix weights to learn and the output of the match function is a scalar. The most complicated and widely adopted match function is based on a fully connected layer with element-wise matching (Mou et al. 2016; Tai, Socher, and Manning 2015; Bowman et al. 2016; 2018; Tay, Tuan, and Hui 2017):

$$\begin{aligned} \mathbf{m} &= \text{Match}(\mathbf{h}^{\text{s}}, \mathbf{h}^{\text{t}}), \\ &= g\left(\text{Concat}\left(\mathbf{h}^{\text{s}}, \mathbf{h}^{\text{t}}, \mathbf{h}^{\text{s}} - \mathbf{h}^{\text{t}}, \mathbf{h}^{\text{s}} * \mathbf{h}^{\text{t}}\right)\right), \end{aligned} \tag{11}$$

where the function $g(\cdot)$ is a non-linear transformation with ReLU as activation function and output of this function is a vector.

Next, we also explore two **aggregation functions**, where the inputs are the different head-wise matching representations, $\mathbf{m}_1, \mathbf{m}_2 ... \mathbf{m}_{\text{I}}$. The most efficient way is the max pooling:

$$\begin{aligned} \mathbf{e} &= \text{MaxPooling}(\mathbf{m}_1, \mathbf{m}_2 ... \mathbf{m}_{\text{I}}), \\ \mathbf{o} &= \text{ReLU}(\mathbf{e}\mathbf{W}^{\text{e}} + \mathbf{b}^{\text{e}}), \end{aligned} \tag{12}$$

where $\mathbf{W}^{\text{e}} \in \mathbb{R}^{d \times d}$ and $\mathbf{b}^{\text{e}} \in \mathbb{R}^{d}$ are the parameters to learn. And the other way is to directly concatenate them and make use of another non-linear transformation layer to map it into a smaller vector representation:

$$\begin{aligned} \mathbf{e} &= \text{Concat}(\mathbf{m}_1, \mathbf{m}_2 ... \mathbf{m}_{\text{I}}), \\ \mathbf{o} &= \text{ReLU}(\mathbf{e}\mathbf{W}^{\text{c}} + \mathbf{b}^{\text{c}}), \end{aligned} \tag{13}$$

where $\mathbf{W}^{\text{c}} \in \mathbb{R}^{dI \times d}$ and $\mathbf{b}^{\text{c}} \in \mathbb{R}^{d}$ are the parameters to learn. The transformation matrix $\mathbf{W}^{\text{c}}$ here is much larger than $\mathbf{W}^{\text{e}}$.

Loss Function

We will mainly focus on the tasks of sequence pair classification in the paper. The matching representation o will be used as the input of final loss as follows:

$$\text{loss} = -\text{log}\frac{\exp(\mathbf{o} \cdot \mathbf{w}^{\text{label}} + b^{\text{label}})}{\sum_f \exp(\mathbf{o} \cdot \mathbf{w}^{\text{f}} + b^{\text{f}})}, \tag{14}$$

where $\mathbf{w}^{\text{f}} \in \mathbb{R}^{d}$ and $\mathbf{b}^{\text{f}} \in \mathbb{R}$ are the parameters to learn.

Experiments

This section introduces our experiment results, implementation details and further analysis.

Sentence vector-based models	SNLI	MNLI-m	MNLI-mm	QQP	SQuAD-binary
LSTM (Bowman et al. 2015)	80.6	-	-	-	-
SPINN-PI (Bowman et al. 2016)	83.2	-	-	-	-
NSE (Munkhdalai and Yu 2017)	84.6	-	-	-	-
CAFE (Tay, Tuan, and Hui 2017)	85.9	-	-	-	-
RSN (Shen et al. 2018)	86.3	-	-	-	-
DSA (Yoon, Lee, and Lee 2018)	87.4	-	-	-	-
BiLSTM (Wang et al. 2018)	-	70.3	70.8	61.4/81.7	-
BiLSTM+Cove (McCann et al. 2017)	-	64.5	64.8	59.4/83.3	-
BiLSTM+ELMo (Peters et al. 2018)	-	72.9	73.4	65.6/85.7	-
Skip-Thought (Kiros et al. 2015)	-	62.9	62.8	56.4/82.2	-
InferSent (Conneau et al. 2017)	-	58.7	59.1	59.1/81.7	-
GenSen (Subramanian et al. 2018)	-	71.4	71.3	59.8/82.9	-
Classic Match	83.2	73.8	74.2	63.3/85.9	61.5
Single-level Head-wise Match	87.0	77.7	77.4	67.8/88.1	62.1
Multi-level Head-wise Match	**88.1**	**79.2**	**79.3**	**69.0/88.6**	**62.9**
SOTA (cross sentence attention)	91.1	86.7	86.0	72.4/89.6	83.2

Table 2: Experiment results. We only compare with the sentence vector-based models as listed in the SNLI Leaderboard. The results on MNLI-m, MNLI-mm and QQP are tested through GLUE Leaderboard. SOTA are the state-of-the-art models with cross sentence attention on the datasets.

	train	test	#class
SNLI	549,367	9,824	3
MNLI-m	392,702	9,796	3
MNLI-mm	392,702	9,847	3
QQP	363,870	390,964	2
SQuAD-binary	130,319	11,873	2

Table 3: The statistics of different datasets.

Datasets

We test our models on the tasks of 1) Text Entailment, which is to identify the relation (entailment, contradiction and neural) between a sequence pair, such as the datasets of Stanford Natual Language Inference (SNLI) (Bowman et al. 2015), Multi-Genre Natural Language Inference matched (MNLI-m) and mismatched (MNLI-mm) (Williams, Nangia, and Bowman 2018) [2]; 2). Duplicate Question Detection, which is to identify whether the given question pair is duplicate or not, such as Quora Question Pairs (QQP); 3) Question Answering, such as the binary classification setting of Stanford Question Answering Dataset 2.0 (SQuAD-binary) (Rajpurkar, Jia, and Liang 2018) where we only need to predict whether the given passage can answer the question or not.

We follow the setting of GLUE [3] to split the datasets of MNLI-m, MNLI-mm and QQP. For SQuAD-binary, we use the pulic dev set as test set. The statistics of the number of samples and classes of different datasets is shown in Table 3.

[2]For the setting of MNLI-m, the sequence pairs from training and test sets are derived from the same sources. For the setting of MNLI-mm, the samples from training and test sets are in different genres.

[3]https://gluebenchmark.com/tasks

Experiment Results

Our experiment results are shown in Table 2. For the QQP task, we report the performance of F1 and accuracy. For the other tasks, we only use the accuracy as the evaluation metric.

We compare our head-wise matching based method with a classic way to matching the representations built by encoders. The "Classic Match" is the method making use of the sequence representation built by Transformer, Eqn.(4), and the element-wise match function, Eqn.(11), for sequence pair matching. The "Single-level Head-wise Match" is our method to aggregate the head-wise matching representation by Eqn.(7) with element-wise match function and maxpooling aggregation function. For a fair comparison, we only use the representations in the final layer for sequence matching for both above-mentioned methods. Based on our results, we can clearly see that our head-wise matching based method is significantly better than the classic matching model on all the datasets. We have a further exploration on integrating "Multi-level Head-wise Match" representations, by Eqn.(8), also with element-wise match function and maxpooling aggregation function. We can also see that it's better than only aggregating the head-wise matching representations in a single layer.

We also compare to other sentence vector-based models: LSTM (Bowman et al. 2015) or BiLSTM (Wang et al. 2018) are trained from scratch without special initialization; SPINN-PI (Bowman et al. 2016) is a stack-augmented parser-interpreter neural network; NSE (Munkhdalai and Yu 2017) is based on neural semantic encoder; CAFE (Tay, Tuan, and Hui 2017) is compare, compress and propagate with alignment-factorized encoder; RSN (Shen et al. 2018) is a reinforced self-attention encoder; DSA (Yoon, Lee, and Lee 2018) is a dynamic self-attention encoder. BiLSTM+Cove (McCann et al. 2017) is based a pre-trained encoder on neural ma-

Match (M) and Aggregate (A) Functions	SNLI
Match_cosine (Eqn. 9) + Agg_concate (Eqn. 13)	86.8
Match_bilinear (Eqn. 10) +Agg_concate (Eqn. 13)	86.7
Match_element (Eqn. 11) +Agg_concate (Eqn. 13)	88.0
Match_element (Eqn. 11) +Agg_concate (Eqn. 13) (no hier)	87.6
Match_element (Eqn. 11)+Agg_maxpooling (Eqn. 12)	88.1

Table 4: Comparison of different matching and aggregation functions in the model of Multi-level Head-wise Match. Each matching/aggregate function corresponds to the its actual equation number. "**no hier**" is to simply aggregate all the representations for classification, instead of aggregating all the head representations in each layer and then all the layer presentations.

chine translation. BiLSTM+ELMo (Peters et al. 2018) is based a pre-trained encoder on language modeling. Skip-Thought (Kiros et al. 2015), InferSent (Conneau et al. 2017), GenSen (Subramanian et al. 2018) are also based on pre-trained encoders trained with different methods.

By comparing all the previous models on matching vectorized sequence representations, we can see that our model achieves the best performance under this setting. Although our best models are still worse than state-of-the-art models with cross sentence attention, the performance is quite close on the datasets of SNLI and QQP, both of which have much larger training set than MNLI and SQuAD-binary. For the dataset of SQuAD-binary, all of our models obtain relatively poor performance. One possible reason is the training set of SQuAD-binary is relatively small and therefore insufficient for learning a good encoder. Another main reason is that SQuAD is in paragraph level, which is naturally longer than the sequences in SNLI or QQP. As such, the hidden states of Transformer are still not powerful enough to represent all the information of the paragraphs containing more words.

Implementation Details

Our Transformer is initialized by BERT-large (Devlin et al. 2019) [4], with 24 layers, 16 heads each layer. Each head is a 64 dimentional vector. We limit the maximal single sequence length of SQuAD-binary to 384 and the other datasets 64. We set the batch size to be 16. We tune the learning rate from $[10^{-5}, 2 \times 10^{-5}, 3 \times 10^{-5}]$ and dropout from $[0, 0.1, 0.2, 0.3, 0.4]$. We use a single GPU, Nvidia V100 with 16G memory, for training the models.

Analysis

This section presents a comprehensive ablative and qualitative analysis.

Effect of Match-Aggregate functions We analyze the performance of different match functions and aggregation functions based on the experiments on the SNLI dataset. The experiment results are shown in Table 4. According to our results, we observe "Match_element" performs the best in all of our experiments in Table 2 also rely on this match function. Although the "Match_cosine" function performs marginally worse, it requires fewer parameters and is also faster than "Match_element", serving as a good choice when in large scale settings.

As for the aggregation methods, we can see that "Agg_maxpooling" and "Agg_concate" achieve similar performance. However, "Agg_maxpooling" requires I, the number of heads, times fewer parameters for training. In a similar fashion, we also extend our model with this aggregation method to other datasets. Additionally, we also have a comparison with the models with and without hierarchical aggregation, denoted as the "no hier" line in Table 4. The "no hier" method performs slightly worse at extracting the most useful head-wise match representation for the final classification.

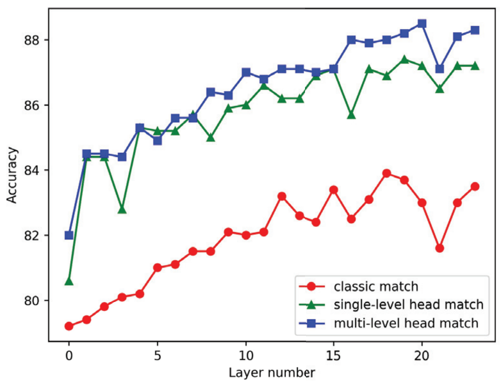


Figure 2: The performance of matching models on Transformer with different number of layers.

Effect of Number of Layers Next, we analyze the effects of the number of Transformer layers on our models, as shown in the Figure 2. We compare three different settings, i.e., "Classic Match", "Single-level Head-wise Match", and "Multi-level Head-wise Match". We observe that the head-wise matching based methods can always achieve better performance than the classic match method, regardless of how many Transformer layers we use. We also observe that the more number of Transformer layers may not always get better performance for all the match models. Lastly, our "Multi-level Head-wise Match" model dynamically integrates all the useful head-wise matching representations from different layers. As a result, this model achieves better performance

[4]https://github.com/huggingface/pytorch-pretrained-BERT

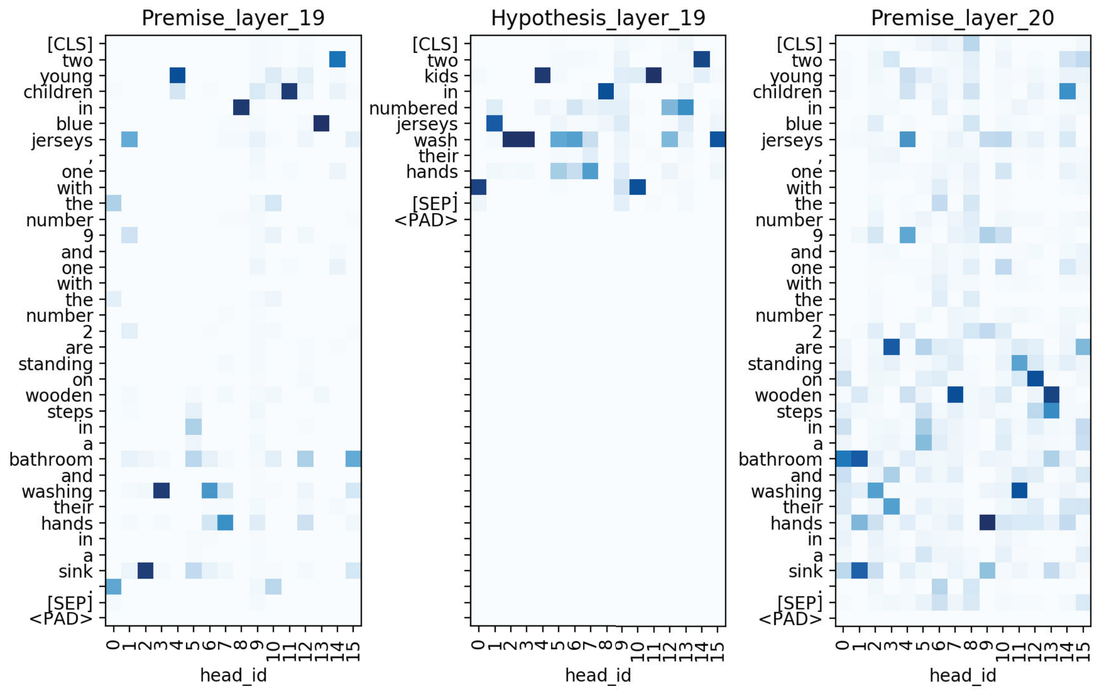


Figure 3: Visualization on the attention weights of different heads (16 in total) from layer 19 and 20 on two sentences.

than only using the representations from the final layer.

Visualization Finally, to further explain our motivation on head-wise match, we conduct a visualization study on the attention weights of Transformer, as shown in Figure 3. We observe the attention weights are not uniformly distributed. For example, the second head in the "Premise_layer_19" focuses on the word "jerseys" and the fifth head pays attention on the word "young". The head with the same id from different encoders can focus on the related words. For example, the second head from "Premise_layer_19" and "Hypothesis_layer_19" focus on the "jerseys", and the forth head focus on the "young" and "kids" respectively. As a result, head-wise matching can help identify which aspects of the hypothesis can be explained by the premise. Moreover, the aggregation of these aspect matching will lead to the final sequence matching.

Additionally, we also observe that some heads pay attention on the punctuations, such as the first head in "Premise_layer_19" and "Hypothesis_layer_19". The maxpooling aggregation function learns to filter these types of head matching that will not contribute to the final prediction. Finally, we can also see that the heads from different layers focus on different aspects of the sequence. For example, the distribution of the words drawn attention by the heads from "Premise_layer_19" and "Premise_layer_20" are quite different. Hence, this explains the need for the head-matching across different layers.

Conclusions

In this paper, we focused on sequence matching based on pre-computed vector representations. We provide a comprehensive deep analysis on the representations built by pretrained Transformer models, making a key observation that merging all the heads to build sequence representation for matching is sub-optimal. Instead, we propose to integrate the head-wise match between sequences, achieving a substantial performance gain on 5 different sequence matching tasks. Moreover, by integrating the head-wise match from all different Transformer layers, our model achieves the best performance among the sentence vector-based models.

References

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Bowman, S. R.; Gauthier, J.; Rastogi, A.; Gupta, R.; Manning, C. D.; and Potts, C. 2016. A fast unified model for parsing and sentence understanding. *acl arXiv preprint arXiv:1603.06021*.

Bowman, S. R.; Pavlick, E.; Grave, E.; Van Durme, B.; Wang, A.; Hula, J.; Xia, P.; Pappagari, R.; McCoy, R. T.; Patel, R.; et al. 2018. Looking for elmo's friends: Sentence-level pretraining beyond language modeling. *arXiv preprint arXiv:1812.10860*.

Conneau, A.; Kiela, D.; Schwenk, H.; Barrault, L.; and Bordes, A. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Das, R.; Dhuliawala, S.; Zaheer, M.; and McCallum, A. 2019. Multi-step retriever-reader interaction for scalable open-domain question answering. In *Proceedings of the International Conference on Learning Representations*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics*.

Dolan, W. B., and Brockett, C. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kim, Y. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kiros, R.; Zhu, Y.; Salakhutdinov, R. R.; Zemel, R.; Urtasun, R.; Torralba, A.; and Fidler, S. 2015. Skip-thought vectors. In *Advances in neural information processing systems*.

Lin, Z.; Feng, M.; Santos, C. N. d.; Yu, M.; Xiang, B.; Zhou, B.; and Bengio, Y. 2017. A structured self-attentive sentence embedding. *Proceedings of the International Conference on Learning Representations*.

McCann, B.; Bradbury, J.; Xiong, C.; and Socher, R. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, 6294–6305.

Mou, L.; Men, R.; Li, G.; Xu, Y.; Zhang, L.; Yan, R.; and Jin, Z. 2016. Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Munkhdalai, T., and Yu, H. 2017. Neural semantic encoders. In *the Conference of the European Chapter of the Association for Computational Linguistics*.

Parikh, A. P.; Täckström, O.; Das, D.; and Uszkoreit, J. 2016. A decomposable attention model for natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics*.

Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. 2018. Improving language understanding by generative pre-training.

Rajpurkar, P.; Jia, R.; and Liang, P. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Reimers, N., and Gurevych, I. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Rocktäschel, T.; Grefenstette, E.; Hermann, K. M.; Kočiskỳ, T.; and Blunsom, P. 2015. Reasoning about entailment with neural attention. In *Proceedings of the International Conference on Learning Representations*.

Seo, M.; Kwiatkowski, T.; Parikh, A. P.; Farhadi, A.; and Hajishirzi, H. 2018. Phrase-indexed question answering: A new challenge for scalable document comprehension. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Shen, T.; Zhou, T.; Long, G.; Jiang, J.; Wang, S.; and Zhang, C. 2018. Reinforced self-attention network: a hybrid of hard and soft attention for sequence modeling. In *Proceedings of International Joint Conference on Artificial Intelligence*.

Subramanian, S.; Trischler, A.; Bengio, Y.; and Pal, C. J. 2018. Learning general purpose distributed sentence representations via large scale multi-task learning. In *Proceedings of the International Conference on Learning Representations*.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the Conference on Association for Computational Linguistics*.

Tan, M.; Santos, C. d.; Xiang, B.; and Zhou, B. 2015. Lstm-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv:1511.04108*.

Tay, Y.; Tuan, L. A.; and Hui, S. C. 2017. Compare, compress and propagate: Enhancing neural architectures with alignment factorization for natural language inference. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Wan, S.; Dras, M.; Dale, R.; and Paris, C. 2006. Using dependency-based features to take the "para-farce" out of paraphrase. In *Proceedings of the Australasian Language Technology Workshop*, volume 2006.

Wang, S., and Jiang, J. 2016. Learning natural language inference with LSTM. In *Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics*.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Williams, A.; Nangia, N.; and Bowman, S. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics*.

Yin, W.; Schütze, H.; Xiang, B.; and Zhou, B. 2015. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*.

Yoon, D.; Lee, D.; and Lee, S. 2018. Dynamic self-attention: Computing attention over words dynamically for sentence embedding. *arXiv prenprint arXiv:1808.07383*.