Simple and Effective Text Matching with Richer Alignment Features

Runqi Yang[1], **Jianhai Zhang**[2], **Xing Gao**[2], **Feng Ji**[2], **Haiqing Chen**[2]
[1]Department of Computer Science and Technology, Nanjing University, China
runqiyang@gmail.com
[2]Alibaba Group, Hangzhou, China
{tanfan.zjh,gaoxing.gx,zhongxiu.jf,
haiqing.chenhq}@alibaba-inc.com

Abstract

In this paper, we present a fast and strong neural approach for general purpose text matching applications. We explore what is sufficient to build a fast and well-performed text matching model and propose to keep three key features available for inter-sequence alignment: original point-wise features, previous aligned features, and contextual features while simplifying all the remaining components. We conduct experiments on four well-studied benchmark datasets across tasks of natural language inference, paraphrase identification and answer selection. The performance of our model is on par with the state-of-the-art on all datasets with much fewer parameters and the inference speed is at least 6 times faster compared with similarly performed ones.

1 Introduction

Text matching is a core research area in natural language processing with a long history. In text matching tasks, a model takes two text sequences as input and predicts a category or a scala value indicating their relationship. A wide range of tasks, including natural language inference (also known as recognizing textual entailment) (Bowman et al., 2015; Khot et al., 2018), paraphrase identification (Wang et al., 2017), answer selection (Yang et al., 2015), and so on, can be seen as specific forms of text matching problems. Research on general purpose text matching algorithm is beneficial to a large number of relevant applications.

Deep neural networks are the most popular choices for text matching nowadays. Semantic alignment and comparison of two text sequences are the keys in neural text matching. Many previous deep neural networks contain a single inter-sequence alignment layer. To make full use of this only alignment process, the model has to take rich external syntactic features or hand-designed alignment features as additional inputs of the alignment layer (Chen et al., 2017; Gong et al., 2018), adopt a complicated alignment mechanism (Wang et al., 2017; Tan et al., 2018), or build a vast amount of post-processing layers to analyze the alignment result (Tay et al., 2018b; Gong et al., 2018).

More powerful models can be built with multiple inter-sequence alignment layers. Instead of making a prediction based on the comparison result of a single alignment process, a stacked model with multiple alignment layers maintains its intermediate states and gradually refines its predictions. However, suffering from inefficient propagation of lower-level features and vanishing gradients, these deeper architectures are harder to train. Recent works have come up with ways of connecting stacked building blocks including dense connection (Tay et al., 2018a; Kim et al., 2018) and recurrent neural networks (Liu et al., 2018), which strengthen the propagation of lower-level features and yield better results than those with a single alignment process.

This paper presents RE2, a fast and strong neural architecture with multiple alignment processes for general purpose text matching. We question the necessity of many slow components in text matching approaches presented in previous literature, including complicated multi-way alignment mechanisms, heavy distillations of alignment results, external syntactic features, or dense connections to connect stacked blocks when the model is going deep. These design choices slow down the model by a large amount and can be replaced by much more lightweight and equally effective ones. Meanwhile, we highlight three key components for an efficient text matching model. These components, which the name RE2 stands for, are previous aligned features (**R**esidual vectors), original point-wise features (**E**mbedding vectors), and contextual features (**E**ncoded vectors). The re-

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4699–4709
Florence, Italy, July 28 - August 2, 2019.

maining components can be as simple as possible to keep the model fast while still yielding strong performance.

The general architecture of RE2 is illustrated in Figure 1. An embedding layer first embeds discrete tokens. Several same-structured blocks consisting of encoding, alignment and fusion layers then process the sequences consecutively. These blocks are connected by an augmented version of residual connections (see section 2.1). A pooling layer aggregates sequential representations into vectors which are finally processed by a prediction layer to give the final prediction. The implementation of each layer is kept as simple as possible, and the whole model, as a well-organized combination, is quite powerful and lightweight at the same time.

Our proposed method achieves the performance on par with the state-of-the-art on four benchmark datasets across three different tasks, namely SNLI and SciTail for natural language inference, Quora Question Pairs for paraphrase identification, and WikiQA for answer selection. Furthermore, our model has the least number of parameters and the fastest inference speed in all similarly-performed models. We also conduct an ablation study to compare with alternative implementations of most components, perform robustness checks to see whether the model is robust to changes of structural hyperparameters, explore what roles the three key features in RE2 play by comparing their occlusion sensitivity and show the evolution of alignment results by a case study. We release the source code[1] of our experiments for reproducibility and hope to facilitate future researches.

2 Our Approach

In this section, we introduce our proposed approach RE2 for text matching. Figure 1 gives an illustration of the overall architecture. Two text sequences are processed symmetrically before the prediction layer, and all parameters except those in the prediction layer are shared between the two sequences. For conciseness, we omit the part for the other sequence in the figure.

In RE2, tokens in each sequence are first embedded by the embedding layer and then processed consecutively by N same-structured blocks with independent parameters (dashed boxes in Figure 1) connected by augmented residual connections. Inside each block, a sequence encoder first computes contextual features of the sequence (solid rectangles in Figure 1). The input and output of the encoder are concatenated and then fed into an alignment layer to model the alignment and interaction between the two sequences. A fusion layer fuses the input and output of the alignment layer. The output of the fusion layer is considered as the output of this block. The output of the last block is sent to the pooling layer and transformed into a fixed-length vector. The prediction layer takes the two vectors as input and predicts the final target. The cross entropy loss is optimized to train the model in classification tasks.

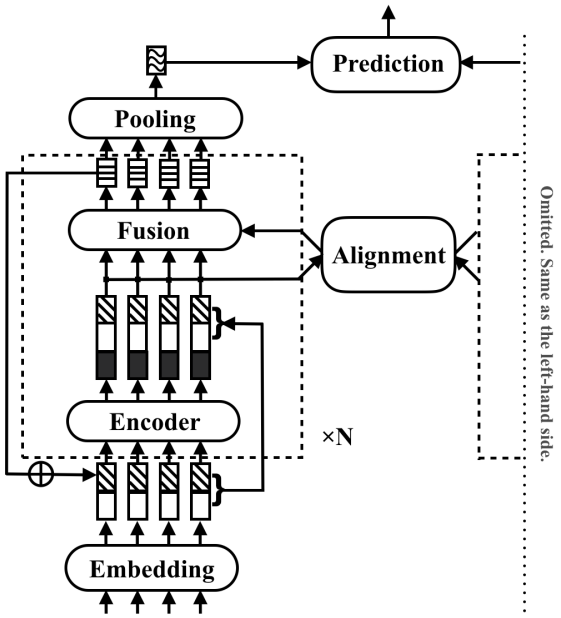


Figure 1: An overview of RE2. There are three parts in the input of alignment and fusion layers: original pointwise features (**E**mbedding vectors, denoted by blank rectangles), previous aligned features (**R**esidual vectors, denoted by rectangles with diagonal stripes), and contextual features (**E**ncoded vectors, denoted by solid rectangles). The architecture on the right is the same as the one on the left so it's omitted for conciseness.

The implementation of each layer is kept as simple as possible. We use only word embeddings in the embedding layer, without character embeddings or syntactic features. Vanilla multi-layer convolutional networks with same padding (Collobert et al., 2011) are adopted as the encoder. Recurrent networks are slower and do not lead to further improvements, so they are not adopted here. A max-over-time pooling operation (Collobert et al., 2011) is used in the pooling layer.

[1] https://github.com/hitvoice/RE2, under the Apache License 2.0.

The details of augmented residual connections and other layers are introduced as follows.

2.1 Augmented Residual Connections

To provide richer features for alignment processes, RE2 adopts an augmented version of residual connections to connect consecutive blocks. For a sequence of length l, We denote the input and output of the n-th block as $x^{(n)} = (x_1^{(n)}, x_2^{(n)}, \ldots, x_l^{(n)})$ and $o^{(n)} = (o_1^{(n)}, o_2^{(n)}, \ldots, o_l^{(n)})$, respectively. Let $o^{(0)}$ be a sequence of zero vectors. The input of the first block $x^{(1)}$, as mentioned before, is the output of the embedding layer (denoted by blank rectangles in Figure 1). The input of the n-th block $x^{(n)}$ ($n \geq 2$), is the concatenation of the input of the first block $x^{(1)}$ and the summation of the output of previous two blocks (denoted by rectangles with diagonal stripes in Figure 1):

$$x_i^{(n)} = [x_i^{(1)}; o_i^{(n-1)} + o_i^{(n-2)}], \tag{1}$$

where $[;]$ denotes the concatenation operation.

With augmented residual connections, there are three parts in the input of alignment and fusion layers, namely original point-wise features kept untouched along the way (**E**mbedding vectors), previous aligned features processed and refined by previous blocks (**R**esidual vectors), and contextual features from the encoder layer (**E**ncoded vectors). Each of these three parts plays a complementing role in the text matching process.

2.2 Alignment Layer

A simple form of alignment based on the attention mechanism is used following Parikh et al. (2016) with minor modifications. The alignment layer, as shown in Figure 1, takes features from the two sequences as input and computes the aligned representations as output. Input from the first sequence of length l_a is denoted as $a = (a_1, a_2, \ldots, a_{l_a})$ and input from the second sequence of length l_b is denoted as $b = (b_1, b_2, \ldots, b_{l_b})$. The similarity score e_{ij} between a_i and b_j is computed as the dot product of the projected vectors:

$$e_{ij} = F(a_i)^T F(b_j). \tag{2}$$

F is an identity function or a single-layer feed-forward network. The choice is treated as a hyper-parameter.

The output vectors a' and b' are computed by weighted summation of representations of the other sequence. The summation is weighted by similarity scores between the current position and the corresponding positions in the other sequence:

$$\begin{aligned} a'_i &= \sum_{j=1}^{l_b} \frac{\exp(e_{ij})}{\sum_{k=1}^{l_b} \exp(e_{ik})} b_j, \\ b'_j &= \sum_{i=1}^{l_a} \frac{\exp(e_{ij})}{\sum_{k=1}^{l_a} \exp(e_{kj})} a_i. \end{aligned} \tag{3}$$

2.3 Fusion Layer

The fusion layer compares local and aligned representations in three perspectives and then fuse them together. The output of the fusion layer for the first sequence $\bar{a}$ is computed by

$$\begin{aligned} \bar{a}_i^1 &= G_1([a_i; a'_i]), \\ \bar{a}_i^2 &= G_2([a_i; a_i - a'_i]), \\ \bar{a}_i^3 &= G_3([a_i; a_i \circ a'_i]), \\ \bar{a}_i &= G([\bar{a}_i^1; \bar{a}_i^2; \bar{a}_i^3]), \end{aligned} \tag{4}$$

where G_1, G_2, G_3, and G are single-layer feed-forward networks with independent parameters and $\circ$ denotes element-wise multiplication. The subtraction operator highlights the difference between the two vectors while the multiplication highlights similarity. Formulations for $\bar{b}$ are similar and omitted here.

2.4 Prediction Layer

The prediction layer takes the vector representations of the two sequences v_1 and v_2 from the pooling layers as input and predicts the final target following Mou et al. (2016):

$$\hat{\mathbf{y}} = H([v_1; v_2; v_1 - v_2; v_1 \circ v_2]). \tag{5}$$

H is a multi-layer feed-forward neural network. In a classification task, $\hat{\mathbf{y}} \in \mathcal{R}^C$ represents the unnormalized predicted scores for all classes where C is the number of classes. The predicted class is $\hat{y} = \text{argmax}_i\, \hat{\mathbf{y}}_i$. In a regression task, $\hat{\mathbf{y}}$ is the predicted scala value.

In symmetric tasks like paraphrase identification, a symmetric version of the prediction layer is used for better generalization:

$$\hat{\mathbf{y}} = H([v_1; v_2; |v_1 - v_2|; v_1 \circ v_2]). \tag{6}$$

We also provide a simplified version of the prediction layer. Which version to use is treated as a hyperparameter. The simplified prediction layer can be expressed as:

$$\hat{\mathbf{y}} = H([v_1; v_2]). \tag{7}$$

3 Experiments

3.1 Datasets

In this section, we briefly introduce datasets used in the experiments and their evaluation metrics.

SNLI (Bowman et al., 2015) (Stanford Natural Language Inference) is a benchmark dataset for natural language inference. In natural language inference tasks, the two input sentences are asymmetrical. The first one is called "premise" and the second is called "hypothesis". The dataset contains 570k human annotated sentence pairs from an image captioning corpus, with labels "entailment", "neutral", "contradiction" and "-". The "-" label indicates that the annotators cannot reach an agreement, so we ignore text pairs with this kind of labels in training and testing following Bowman et al. (2015). We use the same dataset split as in the original paper. Accuracy is used as the evaluation metric for this dataset.

SciTail (Khot et al., 2018) (Science Entailment) is an entailment classification dataset constructed from science questions and answers. Since scientific facts cannot contradict with each other, this dataset contains only two types of labels, entailment and neutral. We use the original dataset partition. This dataset contains 27k examples in total. 10k examples are with entailment labels and the remaining 17k are labeled as neutral. Accuracy is used as the evaluation metric for this dataset.

Quora Question Pairs[2] is a dataset for paraphrase identification with two classes indicating whether one question is a paraphrase of the other. The dataset contains more than 400k real question pairs collected from Quora.com. We use the same dataset partition as mentioned in Wang et al. (2017). Accuracy is used as the evaluation metric for this dataset.

WikiQA (Yang et al., 2015) is a retrieval-based question answering dataset based on Wikipedia. It contains questions and their candidate answers, with binary labels indicating whether a candidate sentence is a correct answer to the question it belongs to. This dataset has 20.4k training pairs, 2.7k development pairs, and 6.2k testing pairs. Mean average precision (MAP) and mean reciprocal rank (MRR) are used as the evaluation metrics for this task.

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

3.2 Implementation Details

We implement our model with TensorFlow (Abadi et al., 2016) and train on Nvidia P100 GPUs. We tokenize sentences with the NLTK toolkit (Bird et al., 2009), convert them to lower cases and remove all punctuations. We do not limit the maximum sequence length, and all sequences in a batch are padded to the batch-wise maximum. Word embeddings are initialized with 840B-300d GloVe word vectors (Pennington et al., 2014) and fixed during training. Embeddings of out-of-vocabulary words are initialized to zeros and fixed as well. All other parameters are initialized with He initialization (He et al., 2015) and normalized by weight normalization (Salimans and Kingma, 2016). Dropout with a keep probability of 0.8 is applied before every fully-connected or convolutional layer. The kernel size of the convolutional encoder is set to 3. The prediction layer is a two-layer feed-forward network. The hidden size is set to 150 in all experiments. Activations in all feed-forward networks are GeLU activations (Hendrycks and Gimpel, 2016), and we use $\sqrt{2}$ as an approximation of the variance balancing parameter for GeLU activations in He initialization. We scale the summation in augmented residual connections by $1/\sqrt{2}$ when $n \geq 3$ to preserve the variance under the assumption that the two addends have the same variance.

The number of blocks is tuned in a range from 1 to 3. The number of layers of the convolutional encoder is tuned from 1 to 3. Although in robustness checks (Table 7) we validate with up to 5 blocks and layers, in all other experiments we deliberately limit the maximum number of blocks and number of layers to 3 to control the size of the model. We use the Adam optimizer (Kingma and Ba, 2015) and an exponentially decaying learning rate with a linear warmup. The initial learning rate is tuned from 0.0001 to 0.003. The batch size is tuned from 64 to 512. The threshold for gradient clipping is set to 5. For all the experiments except for the comparison of ensemble models, we report the average score and the standard deviation of 10 runs.

3.3 Results on Natural Language Inference

Results on SNLI dataset are listed in Table 1. We compare single models and ensemble models. For a fair comparison, we only compare with results obtained without external contextualized embed-

Model	Params	Acc.(%)
DecAtt (Parikh et al., 2016)	0.6M	86.8
BiMPM (Wang et al., 2017)	1.6M	86.9
ESIM (Chen et al., 2017)	4.3M	88.0
DIIN (Gong et al., 2018)	4.4M	88.0
MwAN (Tan et al., 2018)	14M	88.3
CAFE (Tay et al., 2018b)	4.7M	88.5
HIM (Chen et al., 2017)	7.7M	88.6
SAN (Liu et al., 2018)	3.5M	88.6
CSRAN (Tay et al., 2018a)	13.9M	88.7
DRCN (Kim et al., 2018)	6.7M	**88.9**
RE2 (ours)	2.8M	**88.9±0.1**
BiMPM (ensemble)	6.4M	88.8
DIIN (ensemble)	17M	88.9
CAFE (ensemble)	17.5M	89.3
MwAN (ensemble)	58M	89.4
DRCN (ensemble)	53.3M	**90.1**
RE2 (ensemble)	22.4M	89.9

Table 1: Experimental results on SNLI test set.

Model	Acc(%)
ESIM (Chen et al., 2017)	70.6
DecompAtt (Parikh et al., 2016)	72.3
DGEM (Khot et al., 2018)	77.3
HCRN (Tay et al., 2018c)	80.0
CAFE (Tay et al., 2018b)	83.3
CSRAN (Tay et al., 2018a)	**86.7**
RE2 (ours)	**86.0±0.6**

Table 2: Experimental results on SciTail test set.

dings. In the ensemble experiment, we train 8 models with different random seeds and ensemble the results by a voting strategy.

Our method obtains a result on par with the state-of-the-art among single models and a highly competitive result among ensemble models, with only a few parameters. Compared to SAN, our model reduces 20% parameters while improves the performance by 0.3% in accuracy, which indicates that our proposed architecture is highly efficient.

Results on Scitail dataset are listed in Table 2. The performance of our method is very close to state-of-the-art. This dataset is considered much more difficult with fewer training data available and generally low accuracy as a binary classification problem. The variance of the results is larger since the size of training and test set is only 4% and 20% compared to those of SNLI.

3.4 Results on Paraphrase Identification

Results on Quora dataset are listed in Table 3. Since paraphrase identification is a symmetric task where two input sequences can be swapped with no effect to the label of the text pair, in hyperparameter tuning we validate between two symmetric versions of the prediction layer (Equation 6 and Equation 7) and use no additional data augmentation. The performance of RE2 is on par with the state-of-the-art on this dataset.

Model	Acc.(%)
BiMPM (Wang et al., 2017)	88.2
pt-DecAttn-word (Tomar et al., 2017)	87.5
pt-DecAttn-char (Tomar et al., 2017)	88.4
DIIN (Gong et al., 2018)	**89.1**
MwAN (Tan et al., 2018)	**89.1**
CSRAN (Tay et al., 2018a)	**89.2**
SAN (Liu et al., 2018)	**89.4**
RE2 (ours)	**89.2±0.2**

Table 3: Experimental results on Quora test set.

Model	MAP	MRR
ABCNN (Yin et al., 2016)	0.6921	0.7108
KVMN (Miller et al., 2016)	0.7069	0.7265
BiMPM (Wang et al., 2017)	0.718	0.731
IWAN (Shen et al., 2017)	0.733	0.750
CA (Wang and Jiang, 2017)	**0.7433**	**0.7545**
HCRN (Tay et al., 2018c)	**0.743**	**0.756**
RE2 (ours)	**0.7452 ±0.0044**	**0.7618 ±0.0040**

Table 4: Experimental results on WikiQA test set.

3.5 Results on Answer Selection

Results on WikiQA dataset are listed in Table 4. Note that some of the previous methods round their reported results to three decimal points, but we choose to align with the original paper (Yang et al., 2015) and round our results to four decimal points. In hyperparameter tuning, we choose the best hyperparameters including early stopping according to MRR on WikiQA development set. We obtain a result on par with the state-of-the-art reported on this dataset. It's worth mentioning that we still train our model by point-wise binary classification loss, unlike some of the previous methods (including HCRN) which are trained by the pairwise ranking loss. Our method can perform well in the answer selection task without any task-specific modifications.

3.6 Inference Time

To show the efficiency of our proposed model, we compare the inference time with some other models whose code is open-source. Table 5 shows the comparison results. All the compared models are implemented in TensorFlow in the original implementations. The † mark indicates that the model uses POS tags as external syntactic features and the computation time of POS tagging is not included. In our RE2 model, the number of en-

Model	time(s/batch)
BiMPM (Wang et al., 2017)	0.05 ± 0.00
CAFE† (Tay et al., 2018b)	0.07 ± 0.01
DIIN† (Gong et al., 2018)	0.85 ± 0.11
DIIN with EM feature†	1.79 ± 0.22
CSRAN† (Tay et al., 2018a)	0.28 ± 0.02
RE2 (1 block)	0.03 ± 0.00
RE2 (2 blocks)	0.04 ± 0.00
RE2 (3 blocks)	0.05 ± 0.00

Table 5: Inference time when batch size = 8 on Intel Core i7 CPUs. Models with † marks use POS tags as external syntactic features and the computation time of POS tagging is not included.

coder layers is set to 3, the largest possible number in all previously reported experiments. Besides, since all the reported results of our proposed method are obtained with no more than 3 blocks, we only measure the inference time of RE2 with 1-3 blocks. We train all the compared models using the official training code and commands released by the authors on Nvidia P100 GPUs and save model checkpoints to disk. After training, all the models are required to make predictions for a batch of 8 pairs of sentences on a MacBook Pro with Intel Core i7 CPUs. The lengths of these sentences are 20 and the maximum number of characters in a word is 12. The reported statistics are the average and the standard deviation of processing 100 batches.

The comparison results in Table 5 show that our method has very high CPU inference speed, even with multiple stacked blocks. Compared with similarly performed methods, ours is 6 times faster than CSRAN and at least 17 times faster than DIIN. With the highly efficient design, our method can perform well without any strong but slow building blocks like recurrent neural networks, dense connections or any syntactic features. Compared with models of similar inference speed, BiMPM and CAFE, ours obtains much higher prediction scores according to Table 1, Table 2, Table 3 and Table 4.

In summary, our proposed method achieves performance on par with the state-of-the-art on all four well-studied datasets across three different tasks with only a few parameters and fast inference speed.

3.7 Analysis

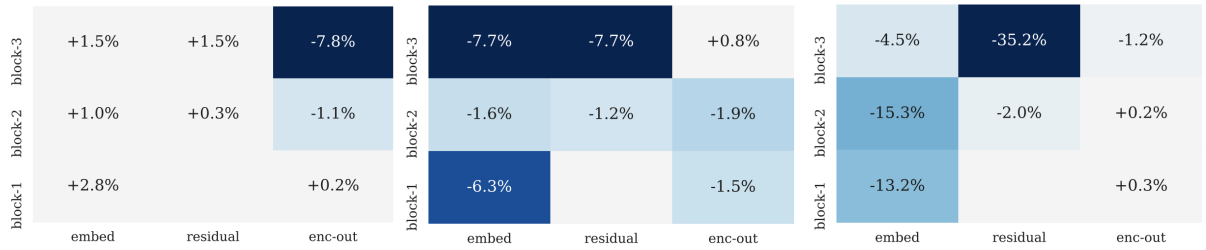
Ablation study. We present an ablation study of our model, comparing the original model with 6 ablation baselines: (1) "w/o enc-in": use directly the output of the encoder as the input of the alignment and fusion layers like in most previous approaches without concatenating the encoder input; (2) "residual conn.": use vanilla residual connections ($x_i^{(n)} = o_i^{(n-1)} + o_i^{(n-2)}$) in place of the augmented version; (3) "simple fusion": use simply $\bar{a}_i = G_1([a_i; a'_i])$ and $\bar{b}_i = G_1([b_i; b'_i])$ as the fusion layer; (4) "alignment alt.": use the alternative version of the alignment layer where F in Equation 2 is a single-layer feed-forward network or an identity function; (5) "prediction alt.": use the alternative version (Equation 5/6 or Equation 7) of the prediction layer; (6) parallel blocks: feed the embeddings directly to all the blocks and sum up their outputs as the input of the pooling layer instead of processing input sequences consecutively by each block. The last setting is designed to study whether the improvement is due to deeper architecture or just a larger amount of parameters.

	SNLI	Quora	Scitail	WikiQA
original	88.9	89.4	88.9	0.7740
w/o enc-in	87.2	85.7	78.1	0.7146
residual conn.	88.9	89.2	87.4	0.7640
simple fusion	88.8	88.3	87.5	0.7345
alignment alt.	88.7	89.3	88.2	0.7702
prediction alt.	88.9	89.2	88.8	0.7558
parallel blocks	88.8	88.6	87.6	0.7607

Table 6: Ablation study on dev sets of the corresponding datasets.

The ablation study is conducted on the development set of SNLI, Quora, Scitail, and WikiQA. In WikiQA we choose MRR as the evaluation metric. Note that on SciTail, F in Equation 2 in alignment layers is an identity function while on all other datasets F is a single-layer feed-forward network. On WikiQA, the simplified version (Equation 7) is used as the prediction layer while on all other datasets the full version (Equation 5 or 6) is used. The reported results are the average of 10 runs and the standard deviations are omitted for clarity.

The result is shown in Table 6. The first ablation baseline shows that without richer features as the alignment input, the performance on all datasets degrades significantly. This is the key component in the whole model. The results of the second baseline show that vanilla residual connections without direct access to the original pointwise features are not enough to model the relations in many text matching tasks. The simpler implementation of the fusion layer leads to evidently worse performance, indicating that the fu-



(a) entailment (b) neutral (c) contradiction

Figure 2: Occlusion sensitivity of different parts in the input of the alignment layers on SNLI dev set: original point-wise features (embed), aligned features (residual), and contextual features (enc-out).

	SNLI	Quora	Scitail
1 block	88.1±0.1	88.7±0.1	88.3±0.8
2 blocks	88.9±0.2	89.2±0.2	**88.9±0.3**
3 blocks	88.9±0.1	89.4±0.1	88.8±0.5
4 blocks	**89.0±0.1**	**89.5±0.1**	88.7±0.5
5 blocks	89.0±0.2	89.2±0.2	88.5±0.5
1 enc. layer	88.6±0.2	88.9±0.2	88.1±0.4
2 enc. layers	88.9±0.2	89.2±0.2	88.9±0.3
3 enc. layers	**89.2±0.1**	**89.2±0.1**	88.7±0.6
4 enc. layers	89.1±0.0	89.1±0.1	88.7±0.5
5 enc. layers	89.0±0.1	89.0±0.2	**89.1±0.3**

Table 7: Robustness checks on dev sets of the corresponding datasets.

sion layer cannot be further simplified. On the other hand, the alignment layer and the prediction layer can be simplified on some of the datasets. In the last ablation study, we can see that parallel blocks perform worse than stacked blocks, which supports the preference for deeper models over wider ones.

Robustness checks. To check whether our proposed method is robust to different variants of structural hyperparameters, we experiment with (1) the number of blocks varying from 1 to 5 with the number of encoder layers set to 2; (2) the number of encoder layers varying from 1 to 5 with the number of blocks set to 2. Robustness checks are performed on the development set of SNLI, Quora and Scitail. The result is presented in Table 7. We can see in the table that fewer blocks or layers may not be sufficient but adding more blocks or layers than necessary hardly harms the performance. On WikiQA dataset, our method does not seem to be robust to structural hyperparameter changes. Crane (2018) mentions that on WikiQA dataset a neural matching model (Severyn and Moschitti, 2015) trained with different random seeds can result in differences up to 0.08 in MAP and MRR. We leave the further investigation of the high variance on the WikiQA dataset for further work.

Occlusion sensitivity. To better understand what roles the three alignment features play, we perform an analysis of occlusion sensitivity similar to those in computer vision (Zeiler and Fergus, 2014). We use a three-block RE2 model to predict on SNLI dev set, mask one feature in one block to zeros at a time and report changes in accuracy of the three categories: entailment, neutral and contradiction. Occlusion sensitivity can help to reveal how much the model depends on each part when deciding on a specific category and we can make some speculations about how the model works based on the observations. Figure 2 shows the result of occlusion sensitivity. Previous aligned features are absent in the first block and thus left blank.

The text matching process can be abstracted, with moderate simplifications, to three stages: aligning tokens between the two sequences, focusing on a subset of the aligned pairs, discerning the semantic relations between the attended pairs. Each of the three key features in RE2 has a closer connection with one of the stages.

As we can see in Figure 2a, contextual features, represented by the output of the encoder, are indispensable when predicting entailment. These features connect with the first stage of text matching. The sequence encoder, implemented by convolutional networks, models local and phrase-level semantics, which helps to build correct alignment for each position. For example, consider the pair "A red car is next to a green house" and "A red car is parked near a house". If the noun phrases in the two sentences are not correctly modeled by the contextual encoding and "green" is incorrectly aligned with another color word "red", the pair looks much less like entailment.

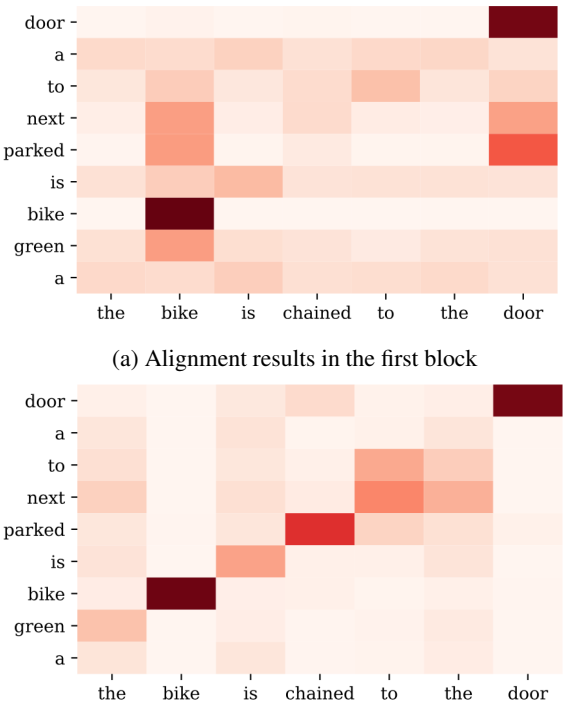
In Figure 2b and Figure 2c, we can see that lacking direct access of previous aligned features

(residual vectors), especially in the final block, results in significant degradation when predicting neutral and contradiction. Previous aligned features are related to the second stage of focusing on a subset of the aligned pairs. Without correct focus, the model may ignore non-entailing pairs and attend to other trivially aligned and semantically matched pairs, which results in failure in predicting neutral and contradiction. The importance of each position can be distilled and stored in previous aligned features and helps the model to focus in latter blocks.

We can conclude from Figure 2b and Figure 2c that when original point-wise features represented by embedding vectors are not directly accessible by alignment layers and fusion layers, the model is struggling to predict neutral and contradiction correctly. Original point-wise features connect with the final stage where semantic differences between aligned pairs are compared. Intact point-wise representations of the aligned pairs facilitate the model in the comparison of their semantic differences, which plays a vital role in predicting neutral and contradiction.

Case study. We present a case study of our model to show how inter-sequence alignment results evolve in our stacked architecture. An example pair of sentences are chosen from the development set of the SNLI dataset. The premise is "A green bike is parked next to a door", and the hypothesis is "The bike is chained to the door". Figure 3 shows the visualization of the attention distribution (normalized e_{ij} in Equation 3) in alignment layers of the first and the last blocks.

In the first block, the alignment results are almost word- or phrase-level. "parked next to" is associated mostly with "bike" and "door" since there is a weaker direct connection between "parked" and "chained". In the final block, the alignment results take consideration of the semantics and structures of the whole sentences. The word "parked" is strongly associated with "chained" and "next to" is aligned with "to the" following "chained". With correct alignment, the model is able to tell that although most parts in the premise entail the aligned parts in the hypothesis, "parked" does not entail "chained", so it correctly predicts that the relation between the two sentences is neutral. Our model keeps the lower-level alignment results as intermediate states and gradually refines them to higher-level ones.



(a) Alignment results in the first block

(b) Alignment results in the third block

Figure 3: A case study of the natural language inference task. The premise is "A green bike is parked next to a door", and the hypothesis is "The bike is chained to the door".

4 Related Work

Deep neural networks are dominant in the text matching area. Semantic alignment and comparison between two text sequences lie in the core of text matching. Early works explore encoding each sequence individually into a vector and then building a neural network classifier upon the two vectors. In this paradigm, recurrent (Bowman et al., 2015), recursive (Tai et al., 2015) and convolutional (Yu et al., 2014; Tan et al., 2016) networks are used as the sequence encoder. The encoding of one sequence is independent of the other in these models, making the final classifier hard to model complex relations.

Later works, therefore, adopt the matching aggregation framework to match two sequences at lower levels and aggregate the results based on the attention mechanism. DecompAtt (Parikh et al., 2016) uses a simple form of attention for alignment and aggregate aligned representations with feed-forward networks. ESIM (Chen et al., 2017) uses a similar attention mechanism but employs bidirectional LSTMs as encoders and aggregators.

Three major paradigms are adopted to further

improve performance. First is to use richer syntactic or hand-designed features. HIM (Chen et al., 2017) uses syntactic parse trees. POS tags are found in many previous works including Tay et al. (2018b) and Gong et al. (2018). The exact match of lemmatized tokens is reported as a powerful binary feature in Gong et al. (2018) and Kim et al. (2018). The second way is adding complexity to the alignment computation. BiMPM (Wang et al., 2017) utilizes an advanced multi-perspective matching operation, and MwAN (Tan et al., 2018) applies multiple heterogeneous attention functions to compute the alignment results. The third way to enhance the model is building heavy post-processing layers for the alignment results. CAFE (Tay et al., 2018b) extracts additional indicators from the alignment process using alignment factorization layers. DIIN (Gong et al., 2018) adopts DenseNet as a deep convolutional feature extractor to distill information from the alignment results.

More effective models can be built if inter-sequence matching is allowed to be performed more than once. CSRAN (Tay et al., 2018a) performs multi-level attention refinement with dense connections among multiple levels. DRCN (Kim et al., 2018) stacks encoding and alignment layers. It concatenates all previously aligned results and has to use an autoencoder to deal with exploding feature spaces. SAN (Liu et al., 2018) utilizes recurrent networks to combine multiple alignment results. This paper also proposes a deep architecture based on a new way to connect consecutive blocks named augmented residual connections, to distill previous aligned information which serves as an important feature for text matching.

5 Conclusion

We propose a highly efficient approach, RE2, for general purpose text matching. It achieves the performance on par with the state-of-the-art on four well-studied datasets across three different text matching tasks with only a small number of parameters and very high inference speed. It highlights three key features, namely previous aligned features, original point-wise features, and contextual features for inter-sequence alignment and simplifies most of the other components. Due to its fast speed and strong performance, the model is quite suitable for a wide range of related applications.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA. USENIX Association.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: Analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced LSTM for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1657–1668, Vancouver, Canada. Association for Computational Linguistics.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537.

Matt Crane. 2018. Questionable answers in question answering research: Reproducibility and variability of published results. *Transactions of the Association for Computational Linguistics*, 6:241–252.

Yichen Gong, Heng Luo, and Jian Zhang. 2018. Natural language inference over interaction space. In *Proceedings of the 6th International Conference on Learning Representations*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA. IEEE Computer Society.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with Gaussian error linear units. *Computing Research Repository*, arXiv:1606.08415. Version 3.

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2018. SciTail: A textual entailment dataset from science question answering. In *Proceedings of AAAI*.

Seonhoon Kim, Jin-Hyuk Hong, Inho Kang, and Nojun Kwak. 2018. Semantic sentence matching with densely-connected recurrent and co-attentive information. *Computing Research Repository*, arXiv:1805.11360. Version 2.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

Xiaodong Liu, Kevin Duh, and Jianfeng Gao. 2018. Stochastic answer networks for natural language inference. *Computing Research Repository*, arXiv:1804.07888.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1400–1409, Austin, Texas. Association for Computational Linguistics.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. 2016. Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 130–136, Berlin, Germany. Association for Computational Linguistics.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Tim Salimans and Diederik P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909.

Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 373–382, New York, NY, USA. ACM.

Gehui Shen, Yunlun Yang, and Zhi-Hong Deng. 2017. Inter-weighted alignment network for sentence pair modeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1179–1189, Copenhagen, Denmark. Association for Computational Linguistics.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Chuanqi Tan, Furu Wei, Wenhui Wang, Weifeng Lv, and Ming Zhou. 2018. Multiway attention networks for modeling sentence pairs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4411–4417. International Joint Conferences on Artificial Intelligence Organization.

Ming Tan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2016. Improved representation learning for question answer matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 464–473, Berlin, Germany. Association for Computational Linguistics.

Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. 2018a. Co-stack residual affinity networks with multi-level attention refinement for matching text sequences. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4492–4502, Brussels, Belgium. Association for Computational Linguistics.

Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. 2018b. Compare, compress and propagate: Enhancing neural architectures with alignment factorization for natural language inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1565–1575, Brussels, Belgium. Association for Computational Linguistics.

Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. 2018c. Hermitian co-attention networks for text matching in asymmetrical domains. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4425–4431. International Joint Conferences on Artificial Intelligence Organization.

Gaurav Singh Tomar, Thyago Duque, Oscar Täckström, Jakob Uszkoreit, and Dipanjan Das. 2017. Neural paraphrase identification of questions with noisy pretraining. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 142–147, Copenhagen, Denmark. Association for Computational Linguistics.

Shuohang Wang and Jing Jiang. 2017. A compare-aggregate model for matching text sequences. In *Proceedings of the 5th International Conference on Learning Representations*.

Zhiguo Wang, Wael Hamza, and Radu Florian. 2017. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4144–4150.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, Lisbon, Portugal. Association for Computational Linguistics.

Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. 2016. ABCNN: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*, 4:259–272.

Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. 2014. Deep learning for answer sentence selection. In *NIPS Deep Learning and Representation Learning Workshop, Montreal*.

Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833.