The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Cross Temporal Recurrent Networks for Ranking Question Answer Pairs

Yi Tay,[1] **Luu Anh Tuan,**[2] **Siu Cheung Hui**[3]
[1,3] Nanyang Technological University
School of Computer Science and Engineering, Singapore
[2] Institute for Infocomm Research, Singapore

Abstract

Temporal gates play a significant role in modern recurrent-based neural encoders, enabling fine-grained control over recursive compositional operations over time. In recurrent models such as the long short-term memory (LSTM), temporal gates control the amount of information retained or discarded over time, not only playing an important role in influencing the learned representations but also serving as a protection against vanishing gradients. This paper explores the idea of learning temporal gates for sequence pairs (question and answer), jointly influencing the learned representations in a pairwise manner. In our approach, temporal gates are learned via 1D convolutional layers and then subsequently cross applied across question and answer for joint learning. Empirically, we show that this conceptually simple sharing of temporal gates can lead to competitive performance across multiple benchmarks. Intuitively, what our network achieves can be interpreted as learning representations of question and answer pairs that are aware of what each other is remembering or forgetting, i.e., pairwise temporal gating. Via extensive experiments, we show that our proposed model achieves state-of-the-art performance on two community-based QA datasets and competitive performance on one factoid-based QA dataset.

Introduction

Learning-to-rank for QA (question answering) is a long standing problem in NLP and IR research which benefits a wide assortment of subtasks such as community-based question answering (CQA) and factoid based question answering. The problem is mainly concerned with computing relevance scores between questions and prospective answers and subsequently ranking them. Across the rich history of answer or document retrieval, statistical approaches based on feature engineering are commonly adopted. These models are largely based on complex lexical and syntactic features (Wang and Manning 2010; Zhou et al. 2011; Wang, Ming, and Chua 2009) and a learning-to-rank classifier such as Support Vector Machine (SVM) (Severyn et al. 2014; Filice et al. 2016).

Today, we see a shift into neural question answering. Specifically, end-to-end deep neural networks are used for both automatically learning features and scoring of QA pairs. Popular neural encoders for neural question answering include long short-term memory (LSTM) networks (Hochreiter and Schmidhuber 1997) and convolutional neural networks (CNN). The key idea behind neural encoders is to learn to compose (Li et al. 2015), i.e., compressing an entire sentence into a single feature vector.

While it is possible to encode questions and answers independently, and later merge them with multi-layer perceptrons (MLP) (Severyn and Moschitti 2015), tensor layers (Qiu and Huang 2015) or holographic layers (Tay et al. 2017), it would be desirable for question and answer pairs to benefit from information available from their partner. There have been many models proposed for doing so which adopt techniques for jointly learning question and answer representations. Many of these recent techniques adopt soft-attention matching (Yang et al. 2016; Santos et al. 2016; Zhang et al. 2017) to learn attention weights that are jointly influenced by both question and answer. Subsequently, the joint attention weights are applied accordingly to learn a final representation of question and answer. Performance results have shown that incorporating the interactions between QA pairs can indeed improve the performance of QA systems.

Temporal gates form the cornerstone of modern recurrent neural encoders such as long short-term memory (LSTM) or gated recurrent units (GRU), serving as one of the key mitigation strategies against vanishing gradients. In these models, temporal gates control the inner recursive loop along with the amount of information being discarded and retained at each time step, allowing fine-grained control over the semantic compositionality of learned representations. Our work explores the idea of jointly learning temporal gates for sequence pairs, aiming to learn fine-grained representations of QA pairs which benefit from information pertaining to what each other is remembering or forgetting.

The key idea here is as follows: *By exploiting information about the question, can we learn an optimal way to semantically compose the answer?* (and vice versa). First, consider the following example in Table 1 which highlights the importance of semantic compositionality.

First, it would be easy for many soft-attention and matching-based models to classify this question and answer pair with a high relevance score due to the underlined words

Q:	What deep learning framework should I learn if I want to **get into** deep learning? I am a **beginner** without programming experience. Want to build cool apps.
A:	Tensorflow. It is a pretty solid and low level deep learning framework for **advanced research**.

Table 1: Example of a question and answer pair. Ground truth is negative.

('deep learning' and 'tensorflow'). However, the reason why this is a negative example is in the intricate details which can be effectively learned only via semantic compositionality (e.g., *'without programming experience'*, *'get into'*). On the other hand, by exploiting joint temporal gates, our method learns to compose the sentence given the information about its partner. For instance, without the knowledge of the answer which contains the phrase *'advanced research'*, the question encoder will not know if it should retain the word *'beginner'*. Hence, joint learning of temporal gates can help our model *learn to compose*, by influencing what it remembers and forgets. As a result, this additional knowledge can allow the words in boldface ('*beginner*','*advanced research*') to be strongly retained in the final representation. This is in similar spirit to neural attention. However, our approach jointly learns to compose instead of learning to attend. The difference is at the level which representations are influenced at.

Our Contributions

The main contributions of this work are:

- We introduce a new method for using temporal gates to synchronously and jointly learn the interactions between text pairs. In the context of question answering, we learn which information to remember or discard in the answer while being aware of the context of the question. To the best of our knowledge, this is the first work that performs question answer matching at the temporal gate level.
- We propose a novel neural architecture for QA ranking. Our proposed Cross Temporal Recurrent Network (CTRN) model is largely inspired by the recently incepted Quasi Recurrent Neural Network (QRNN) (Bradbury et al. 2016) and can be considered as a natural extension of QRNN to sequence pairs. Our model takes after QRNN in the sense that gates are first learned (via 1D convolutional layers) and then subsequently applied to temporally adjust the representations. Hence, the facilitation of information flow can be interpreted as *joint pairwise gating*.
- Our proposed CTRN model achieves state-of-the-art performance on two community-based QA (CQA) datasets, namely the Yahoo Answers dataset and the QatarLiving dataset from SemEval 2016. Moreover, our model also achieves highly competitive performance on the TrecQA dataset for factoid based QA. Experimental results show that CTRN outperforms models that utilize attention-based matching while being significantly more efficient. Experimental results also confirm that CTRN improves the underlying QRNN model.

Related Work

This section introduces prior work in the field of neural QA ranking. We also introduce the Quasi Recurrent Neural Network (QRNN) model, which lives at the heart of our proposed approach.

Neural Question Answer Ranking

Convolutional neural network (CNN) (Hu et al. 2014; Severyn and Moschitti 2015) and recurrent models like the long short-term memory (LSTM) (Wang and Nyberg 2015) network are popular neural encoders for the QA ranking problem. (Yu et al. 2014) proposed to use CNN for learning features and subsequently apply logistic regression for generating QA relevance scores. Subsequently, an end-to-end neural architecture based on CNN and bilinear matching was proposed in (Severyn and Moschitti 2015).

In many recent works, the key innovation in most models is the technique used to model interaction between question and answer pairs. The CNN model introduced in (Severyn and Moschitti 2015) uses a MLP to compose vectors of questions and answers. (Qiu and Huang 2015) adopted tensor layers for richer modeling capabilities. (Tay et al. 2017) proposed holographic memory layers. (He, Gimpel, and Lin 2015) proposed Multi-Perspective CNN which matches 'multiple perspectives' based on variations of pooling and convolution schemes. Recent work has showed the effectiveness of learning QA embeddings in non-Euclidean spaces such as Hyperbolic space (Tay, Luu, and Hui 2017). Models based on soft-attention such as Attentive Pooling networks (AP-BiLSTM and AP-CNN) (Santos et al. 2016), AI-CNN (Zhang et al. 2017) and aNMM (attention-based neural matching) (Yang et al. 2016) have also been proposed. These models learn weighted representations of QA pairs using similarity matrix based attentions.

Quasi Recurrent Neural Network (QRNN)

In this section, we introduce Quasi Recurrent Neural Network (Bradbury et al. 2016) and our key motivation of basing CTRN on QRNN. QRNN is a recurrent neural network that is actually a convolutional neural network in disguise. The key intuition with QRNN is that it first learns temporal gates via 1D convolutions and subsequently applies them sequentially. Contrastively, recurrent models learn these gates sequentially. Given an input of a sequence of L vectors $w \in \mathbb{R}^m$ where L is the maximum sequence length and m is the dimension of the vectors, the QRNN model applies three 1D convolution operations as follows:

$$\begin{aligned} \mathbf{Z} &= tanh(\mathbf{W}_z * \mathbf{X}) \\ \mathbf{F} &= \sigma(\mathbf{W}_f * \mathbf{X}) \\ \mathbf{O} &= \sigma(\mathbf{W}_o * \mathbf{X}) \end{aligned} \tag{1}$$

where $\mathbf{X}$ is the input sequence of m dimensional vectors with sequence length L. $\mathbf{W}_x, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{k \times d \times m}$ are parameters of QRNN and $*$ denotes a convolution across the temporal dimension. k is the filter width and d is the output

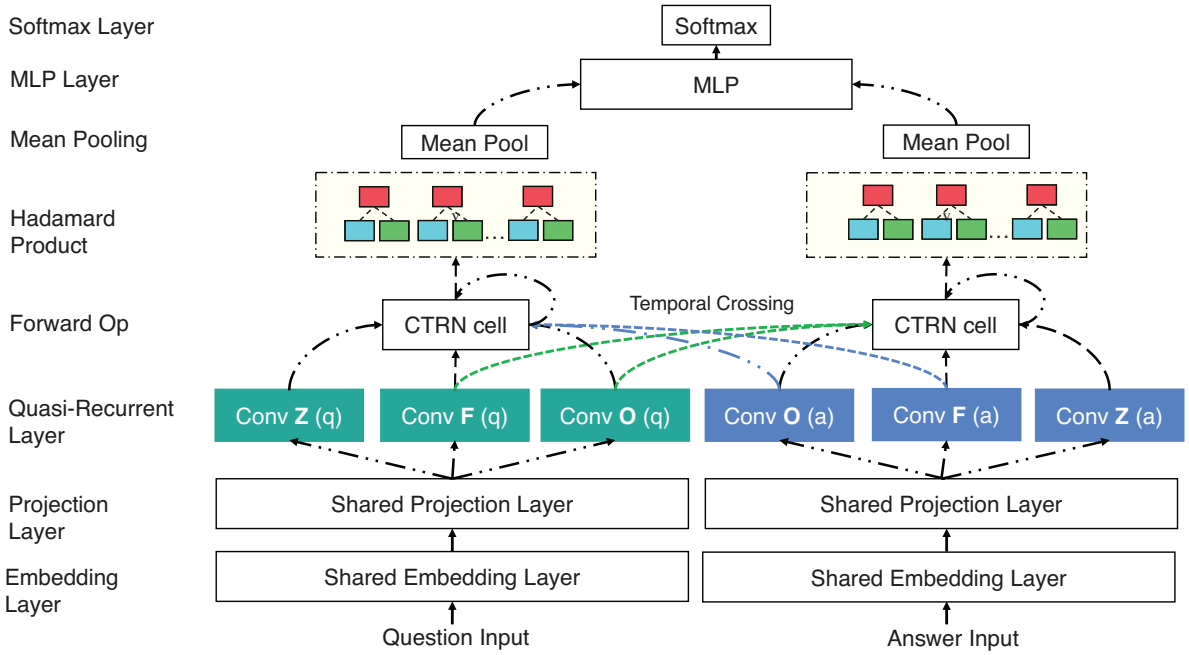

Figure 1: Diagram of our proposed CTRN architecture. Gates (denoted Conv F and Conv O) are prelearned via convolutions and each CTRN cell (for q and a) incorporates the gates of their partner while learning representations. The base representations (in which gates are applied in CTRN) are denoted by Conv Z. Green denotes information flow from question and blue denotes information flow from answer.

dimension. Subsequently, the following equations describe the forward (recursive) operation of the QRNN cell:

$$c_t = f_t \odot c_{t-1} + (1 - f_t) \odot z_t$$
$$h_t = o_t \odot c_t$$

where c_t is the cell state and h_t is the hidden state. f_t, o_t are the forget and output gates respectively at time step t. σ is the sigmoid activation which nonlinearly projects each element of its input to $[0, 1]$. $\mathbf{Z}$ can be regarded as the *convolved* base representation similar to what a traditional CNN model learns. $\mathbf{F}$ and $\mathbf{O}$ are then applied recursively to temporally adjust and influence the semantic compositionality of $\mathbf{Z}$. As such, this makes it *'quasi-recurrent'*. The key difference between QRNN and recurrent models like LSTM is that gates are *prelearned* via convolution while RNN models like LSTM learn their gates sequentially during the recursive forward operation. In short, the forward operation in QRNN is still sequentially applied but is comparatively much cheaper than traditional LSTM cells since gates are merely applied in the case of QRNN. As such, the parallelization of gate learning improves the speed of QRNN as compared to LSTM. In the original paper, QRNN achieved around 4 times less computational time as compared to LSTM models while achieving similar or better performance. For the sake of brevity, we refer interested readers to (Bradbury et al. 2016) for more details.

Inspired by the computational benefits of QRNN, we adopt it as our base model. Next, we also notice an attractive property of QRNN. In QRNN models, because gates are prelearned, it enables us to align temporal gates between two QRNNs easily. Conversely, considering the fact that questions and answers might not have similar sequence length, trying to sequentially align temporal gates in LSTM models can be extremely cumbersome and inefficient. More importantly, temporal gates of LSTM cells do not have global information, i.e., each step is only aware of all steps that precede it. On the other hand, temporal gates from QRNNs have global information about the entire sequence.

Our Proposed Approach

In this section, we describe our novel deep learning model layer-by-layer. The overall architecture of our model is illustrated in Figure 1. For notational convenience, we denote the subscripts q, a on whether a parameter belongs to question or answer respectively.

Embedding + Projection Layer

Our model accepts two sequences of indices (question and answer inputs) which are passed through an embedding layer (shared between q and a inputs) and returns a sequence of n dimensional vectors. In practice, we initialize and **fix** this layer with pretrained embeddings while connecting to a $n \times m$ projection layer. As such, the output of the embedding + projection layer is a m dimensional vector. Note that this layer is shared between question and answer inputs.

Quasi-Recurrent Layer

The input to the quasi-recurrent layer is a sequence of L vectors $w \in \mathbb{R}^m$ where L is the maximum sequence length.

This layer applies three 1D convolution operations as described in Equation (1). Finally, the outputs of the quasi-recurrent layer are representations or matrices $\{\mathbf{Z}_s, \mathbf{F}_s, \mathbf{O}_s\}$ where $s = \{q, a\}$. Note that up till now, this quasi-recurrent layer remains functionally identical to QRNN.

Lightweight Temporal Crossing (LTC)

In this section, we introduce our novel lightweight temporal crossing (LTC) mechanism which lives at the heart of our CTRN model. Our approach extends upon the QRNN model, we leverage the fact that gates $\mathbf{F}_x, \mathbf{O}_x$ are learned non-sequentially. The key idea is to leverage the information in $\mathbf{F}_q, \mathbf{O}_q$ for $\mathbf{Z}_a$ and vice versa. This information flow is denoted by the green and blue arrows in Figure 1. The outputs of this layer are similar to the LSTM model, i.e., they are a sequence of hidden states $\mathbf{H} \in \mathbb{R}^{L \times d}$ where L is the sequence length and d is the number of filters. At this layer, there are two CTRN cells, namely CTRN-Q and CTRN-A, for question and answer representations respectively.

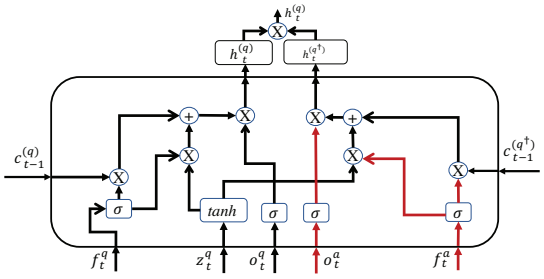

Figure 2: Diagram of a single CTRN-Q cell. Red lines are information flow from answer gates. X denotes element-wise multiplication, $+$ denotes element-wise addition. $tanh$ is the hyperbolic tangent function and σ is the sigmoid function.

In this section, we use CTRN-Q as an example but note that CTRN-A and CTRN-Q are functionally symmetrical. Figure 2 illustrates a single CTRN-Q cell. Each CTRN-Q cell contains two cell states denoted as $c_t^{(q)}$ and $c_t^{(q^\dagger)}$ and two hidden states denoted as $h_t^{(q)}$ and $h_t^{(q^\dagger)}$. As such, there are two learned representations in the CTRN-Q cell, denoted by q and $q^\dagger$ respectively. The first representation is learned as per normal, i.e., applying $\mathbf{F}_q, \mathbf{O}_q$ on $\mathbf{Z}_q$. The second representation is learned by applying partner gates $\mathbf{F}_a$, $\mathbf{O}_a$ on the question representation $\mathbf{Z}_q$. The following equations depict the forward operation of the CTRN-Q cell.

$$
\begin{aligned}
c_t^{(q)} &= f_t^q \odot c_{t-1}^{(q)} + (1 - f_t^q) \odot z_t^q \\
h_t^{(q)} &= o_t^q \odot c_t^{(q)} \\
c_t^{(q^\dagger)} &= f_{t^*}^a \odot c_{t-1}^{(q^\dagger)} + (1 - f_{t^*}^a) \odot z_t^q \\
h_t^{(q^\dagger)} &= o_{t^*}^a \odot c_t^{(q^\dagger)}
\end{aligned}
$$

where f_t^n, o_t^n, z_t^n denote the forget and output gates for text $n \in \{q, a\}$ at time step t. t^* is an aligned time step between question and answer sequences as the sequence length of question and answer might be different. For simplicity, we consider $t^* = t \times \lceil \frac{max(|q|,|a|)}{min(|q|,|a|)} \rceil$. Similarly, the forward operation for CTRN-A is as follows:

$$
\begin{aligned}
c_t^{(a)} &= f_t^a \odot c_{t-1}^{(a)} + (1 - f_t^a) \odot z_t^a \\
h_t^{(a)} &= o_t^a \odot c_t^{(a)} \\
c_t^{(a^\dagger)} &= f_{t^*}^q \odot c_{t-1}^{(a^\dagger)} + (1 - f_{t^*}^q) \odot z_{t^*}^a \\
h_t^{(a^\dagger)} &= o_t^q \odot c_t^{(a^\dagger)}
\end{aligned}
$$

Finally, to obtain a single representation for each question and answer. We simply apply the Hadamard product $\odot$ between hidden states of **each time step**, i.e., $h_t^{(q)} = h_t^{(q)} \odot h_t^{(q^\dagger)}$ and $h_t^{(a)} = h_t^{(a)} \odot h_t^{(a^\dagger)}$. This enables joint representations of temporal gates which form the crux of our LTC mechanism.

Why does this work? Notably, since gates are learned via parameterized convolutional layers, our learned gates ($\mathbf{F}$ and $\mathbf{O}$) not only contain 'local' index-specific information but also 'global' information of the entire text sequence. This is modeled by the parameters of the convolutional layers which produce $\mathbf{F}$ and $\mathbf{O}$. As such, it would suffice to compose them index-wise since the goal is to enable information flow between the temporal gates of question and answer. Our intuition here is to *cross apply* question and answer gates to both question and answer representations so as to enable gradients flow across question and answer during back-propagation. Since the goal is to fuse and not to 'match', we empirically found that soft-attention alignment of gates to yield no performance benefits over a simple index-wise alignment.

Temporal Mean Pooling Layer The output of each CTRN cell is an array of hidden states $[h_1^s, h_2^s..h_L^s]$. In this layer, we apply temporal mean pooling for both CTRN-Q and CTRN-A. The operation of this layer is a simple element-wise average of all output hidden vectors.

Dense Layers (MLP) The inputs of this layer are two vectors which are the final representations of question and answer respectively. In this layer, we concatenate the two vectors and pass them through a series of fully-connected dense layers (or MLP). Likewise, the number of layers is also a hyperparameter to be tuned.

Softmax Layer and Optimization The final output of the hidden layer is then passed through a 2-class softmax layer. The final score of each QA pair is described as follows:

$$
s(q, a) = softmax(W_f\, x + b_f) \tag{2}
$$

where x is the output of the last hidden layer and $W_f \in \mathbb{R}^{h \times 2}$ and $b_f \in \mathbb{R}^2$. h is the size of the hidden layer. Our network minimizes the standard cross entropy loss as its training objective. The choice of a pointwise model is motivated by 1) ease of implementation and 2) previous work (Severyn and Moschitti 2015; Tay et al. 2017; Zhang et al. 2017). The loss function is defined as follows:

$$
L = -\sum_{i=1}^{N} [y_i \log s_i + (1 - y_i) \log(1 - s_i)] + \lambda \left\| \theta \right\|_2^2 \tag{3}
$$

where s is the output of the softmax layer. θ contains all the parameters of the network and $\lambda \|\theta\|_2^2$ is the L2 regularization. The parameters of the network are updated using the Adam Optimizer (Kingma and Ba 2014).

Complexity Analysis

In this section, we study the memory complexity of our model to further justify the *lightweight* aspect in our LTC mechanism. First, our CTRN **does not** incur any parameter cost over the vanilla QRNN model ($3kdm$ for a single QRNN model). As such, the memory complexity and parameter size remain equal to QRNN. This is easy to see as there is no additional parameters added since our model, from a computational graph perspective, is simply adding connections between nodes. Next, we consider the runtime complexity of our model (forward pass). Let d be the number of filters of the convolution layer and L be the maximum sequence length. The computational complexity of a single QRNN cell is $\mathcal{O}(dL)$ excluding convolution operations used to generate $\{\mathbf{F}, \mathbf{O}, \mathbf{Z}\}$. Though the number of operations is approximately doubled (due to cross applying gates), the complexity of a CTRN cell is still $\mathcal{O}(dL)$, i.e., our model still runs in linear time as compared to LSTM models with quadratic time complexity. Overall, our model, though seemingly more complicated, does not increase the parameter size and only incurs a slight increase in computational cost as compared to the already efficient QRNN model. We are also able to leverage the computational benefits of QRNN over the vanilla LSTM model. Table 2 shows a simple comparison of our proposed CTRN model against the standard LSTM and AP-BiLSTM models. We observe that QRNN and CTRN are much more parameter efficient as compared to recurrent models, only taking up $\approx 58\%$ the parameter size of the vanilla LSTM model and being 400% smaller than AP-BiLSTM.

Model	# Mem Complexity	# Params
LSTM	$4(md + d^2) + \mathbf{2dh + h}$	1.79M
AP-BiLSTM	$4(md + d^2) + \mathbf{4d}^2$	5.86M
QRNN	$3\ kdm + \mathbf{2dh + h}$	1.05M
CTRN	$3\ kdm + \mathbf{2dh + h}$	1.05M

Table 2: Memory complexity analysis with shared parameters for q and a. m is the size of the input embeddings. d is the number of filters and the dimensionality of the LSTM model. h is the size of the hidden layer. The complexity highighted in boldface is used to compose q and a. # Params gives an estimate with $d = 512$, $m = 300$, $h = 128$ and $k = 2$. Word embedding parameters are excluded from comparison.

Experiments

To ascertain the effectiveness of our proposed approach, we conduct experiments on three popular benchmark datasets.

Experimental Setup

This section describes the datasets used, baselines compared and evaluation metrics.

Datasets We select three popular benchmark datasets which are described as follows:

- **YahooQA** - Yahoo Answers is a CQA platform. This is a moderately large dataset containing $142,627$ QA pairs which are obtained from the CQA platform. More specifically, preprocessing and testing splits[1] are obtained from (Tay et al. 2017). In their setting, questions and answers that are not in the range of $5 - 50$ tokens are filtered. Additionally, 4 negative samples are generated for each question by sampling from the top 1000 hits using *Lucene* search.
- **QatarLiving** - This is another CQA dataset which was obtained from the popular SemEval-2016 Task 3 Subtask A (CQA). This is a real world dataset obtained from Qatar Living Forums. In this dataset, there are ten answers per thread (question) which are marked as 'Good', 'Potentially Useful' or 'Bad'. Following (Zhang et al. 2017), we treat 'Good' as positive and anything else as negative labels.
- **TrecQA** - This is a popular QA ranking benchmark obtained from the TREC QA Tracks 8-13. QA pairs are generally short and factoid-based consisting trivia like questions. In this dataset, there are **two** training sets, namely TRAIN and TRAIN-ALL. TRAIN consists of QA pairs that have been manually judged and annotated. TRAIN-ALL is an automatically judged dataset of QA pairs and contains a larger number of QA pairs. TRAIN-ALL, being a larger dataset, also contains more noise. Nevertheless, both datasets enable the comparison of all models with respect to the availability and volume of training samples.

The statistics of all datasets, i.e., training sets, development sets and testing sets, are given in Table 3.

	CQA		TrecQA	
	YahooQA	QL	TRAIN	TRAIN-ALL
Train Qns	50.1K	4.8K	94	1229
Dev Qns	6.2K	224	82	82
Test Qns	6.2K	327	100	100
Train Pairs	253K	36K	4.7K	53K
Dev Pairs	31.7K	2.4K	1.1K	1.1K
Test Pairs	31.7K	3.2K	1.5K	1.5K

Table 3: Statistics of datasets. QL denotes the QatarLiving dataset. TRAIN and TRAIN-ALL are two settings of the TrecQA dataset.

Evaluation Metrics For each dataset, we adopt the evaluation metrics used in prior work. For YahooQA, we follow (Tay et al. 2017) that uses P@1 (Precision@1) and MRR (Mean Reciprocal Rank). For QatarLiving, we follow (Zhang et al. 2017) and evaluate on P@1 and MAP (Mean Average Precision). For TrecQA, we follow the experiment procedure in (Severyn and Moschitti 2015) using the official evaluation metrics of MAP and MRR. Since the evaluation

[1]Splits be obtained at https://github.com/vanzytay/YahooQA_Splits.

metrics are commonplace in ranking tasks, we omit any further details for the sake of brevity.

Implementation Details and Baselines For our CTRN model, we tune the output dimension (number of filters) within $[128, 1024]$ in multiples of 128. A single layered CTRN and QRNN is used. The number of dense (MLP) layers is tuned from $[1, 3]$ and learning rate tuned amongst $\{10^{-3}, 10^{-4}, 10^{-5}\}$. Batch size is tuned amongst $\{64, 128, 256, 512\}$. Dropout is set to 0.5 and L2 regularization is set to 4×10^{-6}. Word embedding matrices are all non-trainable and are learned by the projection layer instead. For the three datasets, we adopt dataset-specific baselines largely based on prior published works.

- **YahooQA** - We compare against multiple state-of-the-art models. Specifically, we compare our model with the vanilla LSTM, vanilla CNN, CNTN, NTN-LSTM and HD-LSTM. Since we use the same testing splits, we report the results directly from (Tay et al. 2017). Additionally, we include additional baselines such as AP-BiLSTM and AP-CNN (Santos et al. 2016) which serve as a representative for soft-attention alignment based models. Cosine similarity with pairwise ranking is used as the metric for AP-BiLSTM and AP-CNN following the original implementation. Models superscripted with † are implemented by us. We initialize our model with pretrained GloVE embeddings (Pennington, Socher, and Manning 2014) of $d = 300$.
- **QatarLiving** - The key competitors of this dataset are the CNN-based ARC-I/II architecture by Hu et al. (Hu et al. 2014), the Attentive Pooling CNN (Santos et al. 2016), Kelp (Filice et al. 2016) a feature engineering based SVM method, ConvKN (Barrón-Cedeño et al. 2016) a combination of convolutional tree kernels with CNN and finally AI-CNN (Attentive Interactive CNN) (Zhang et al. 2017), a tensor-based attentive pooling neural model. We initialize with pretrained GloVE embeddings of $d = 200$ trained using the domain-specific unannotated corpus provided by the task.
- **TrecQA** - We compare against published works which include both traditional models and neural models. Moreover, we compare with models reported in (Tay et al. 2017) on TRAIN and TRAIN-ALL datasets to observe the effect of different dataset sizes. The evaluation procedure follows (Severyn and Moschitti 2015) closely. We initialize the embedding layers with the same pretrained word embeddings of $d = 50$ as (Severyn and Moschitti 2015) for fair comparisons against competitor approaches. These embeddings are trained with the Skip-gram model using the Wikipedia and AQUAINT corpus. *Four* word overlap features are also concatenated before the dense layers following (Severyn and Moschitti 2015). We train our model for 25 epochs for TRAIN and 5 epochs for TRAIN-ALL and report the test score from the best performing model on the development set. Hyperparameters are also tuned on the development set. Early stopping is adopted and training is terminated if the validation performance doesn't improve after 5 epochs.

Experimental Results

In this section, we report some observations pertaining to our empirical results.

Model	P@1	MRR
Random Guess	0.200	0.457
BM-25	0.225	0.493
CNN$^\phi$	0.413	0.632
CNTN$^\phi$	0.465	0.632
LSTM$^\phi$	0.465	0.669
NTN-LSTM$^\phi$	0.545	0.731
HD-LSTM$^\phi$	0.557	0.735
AP-CNN†	0.560	0.726
AP-BiLSTM†	0.568	0.731
QRNN†	<u>0.573</u>	<u>0.736</u>
CTRN (This paper)	**0.601**	**0.755**

Table 4: Experimental results on YahooQA. Models are ranked by P@1. Models marked with ϕ are reported directly from (Tay et al. 2017) while † denotes our own implementation. Best result is in boldface and second best is underlined.

Experimental Results on YahooQA Table 4 reports the experimental results on the YahooQA dataset. Firstly, we observe that our proposed CTRN achieves state-of-the-art performance on this dataset. Notably, we outperform HD-LSTM (Tay et al. 2017) by 4% in terms of P@1 and 2% in terms of MRR. CTRN also outperforms attention based models such as AP-BiLSTM and AP-CNN (Santos et al. 2016) by a considerable margin, i.e., of about $2\% - 3\%$. At this junction, we make several observations about our proposed CTRN model. Firstly, this shows that our LTC mechanism is more effective than soft-attention matching on this dataset. Secondly, the merits of this mechanism can be further observed by the performance difference in the QRNN and CTRN. Our proposed CTRN comfortably outperforms QRNN by $2\%-3\%$ in terms of P@1 and MRR. Surprisingly, we see that a simple baseline QRNN performs quite well on this dataset which outperforms other complex models such as NTN-LSTM and HD-LSTM (Tay et al. 2017).

Experimental Results on QatarLiving Table 5 reports our experimental results on the QatarLiving dataset. Our CTRN model outperforms AI-CNN[2] by 2.5% in terms of P@1 while maintaining similar performance on MRR. The performance of the CTRN model also outperforms the baseline QRNN by 3% on P@1. Similar to the Yahoo QA dataset, we also found that the baseline QRNN performed surprisingly well, i.e., outperforming ConvKN and other CNN based models such as ARC-I and ARC-II. Overall, our proposed approach achieves very competitive results on this dataset.

[2]For fair comparison, we compare against the reported results of AI-CNN that does not use handcrafted features.

Model	P@1	MAP
ARC-I CNN	0.741	0.771
ARC-II CNN	0.753	0.780
AP	0.755	0.771
Kelp	0.751	0.792
ConvKN	0.755	0.777
QRNN	0.758	0.783
AI-CNN	<u>0.763</u>	<u>0.791</u>
CTRN (This paper)	**0.788**	**0.794**

Table 5: Experimental results on the QatarLiving dataset. Best result is in boldface and second best is underlined. CTRN outperforms the complex AI-CNN model.

	TRAIN		TRAIN-ALL	
Model	**MAP**	**MRR**	**MAP**	**MRR**
CNN + LR	0.7058	0.7846	0.7113	0.7846
CNN	0.7000	0.7469	0.7216	0.7899
CNTN	0.7045	0.7562	0.7278	0.7831
LSTM	0.7007	0.7777	0.7350	0.8064
MV-LSTM	0.7077	0.7821	0.7327	0.7940
NTN-LSTM	0.7225	0.7904	0.7364	0.8009
HD-LSTM	<u>0.7520</u>	<u>0.8146</u>	0.7499	0.8153
QRNN	0.7000	0.7859	<u>0.7609</u>	<u>0.8227</u>
CTRN (This paper)	**0.7582**	**0.8233**	**0.7712**	**0.8384**

Table 6: Comparisons of various neural baselines on the TREC QA task on two dataset settings TRAIN and TRAIN-ALL. Competitors (except QRNN and CTRN) are reported from (Tay et al. 2017). Best result is in boldface and second best is underlined.

Model	MAP	MRR
Wang et al. (2007)	0.6029	0.6852
Heilman and Smith (2010)	0.6091	0.6917
Wang and Manning (2010)	0.5951	0.6951
Yao (2013)	0.6307	0.7477
Wang et al. (2015) - BM25	0.6370	0.7076
S & M (2013)	0.6781	0.7358
Yih et al. (2013)	0.7092	0.7700
Yu et al. (2014) - CNN+LR	0.7113	0.7846
Wang et al. (2015) - biLSTM	0.7143	0.7913
S & M. (2015) - CNN	0.7459	0.8078
Yang et al. (2016) - aNMM	0.7495	0.8109
Tay et al. (2017) - HD-LSTM	0.7499	0.8153
Bradbury et al. (2016) - QRNN+MLP†	0.7609	0.8227
He and Lin (2016) - MP-CNN	<u>0.7620</u>	<u>0.8300</u>
CTRN† (TRAIN)	0.7582	0.8233
CTRN† (TRAIN-ALL)	**0.7712**	**0.8384**

Table 7: Performance comparisons of all published works on TREC QA dataset. $S\&M$ is short for Severyn and Moschitti. Best result is in boldface and second best is underlined. The MP-CNN result is reported from (Rao, He, and Lin 2016), using the pointwise model for fair comparison. † denotes results reported by this work.

Results on TrecQA Table 6 reports the results on TRAIN and TRAIN-ALL settings of the TrecQA task. CTRN achieves the top results comparing to the multitude of neural baselines. Notably, the performance of CTRN is about $1\% - 5\%$ better than QRNN. QRNN performs very competitively on the TRAIN-ALL setting but fails in comparison for the TRAIN setting. This might be because QRNN, with three 1D convolutional layers, might overfit on the smaller dataset. However, CTRN performs well on smaller TRAIN as well which hints at possibly some regularizing effect of the LTC mechanism. The performance of the vanilla QRNN model on the TRAIN-ALL setting is also surprisingly competitive, outperforming more complex models such as HD-LSTM and NTN-LSTM.

Table 7 reports the results of the CTRN model against other published competitors. We can see that CTRN outperforms many complex neural architectures such as the aNMM model (Yang et al. 2016), HD-LSTM (Tay et al. 2017) and MP-CNN model (He, Gimpel, and Lin 2015; Rao, He, and Lin 2016).

Runtime Comparison

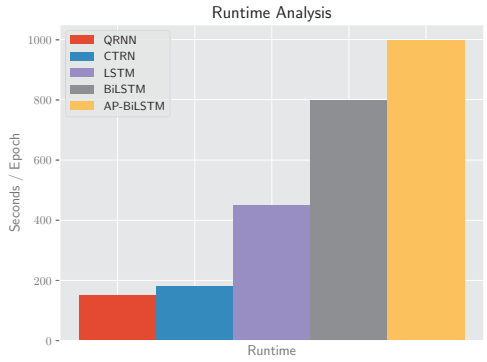

Figure 3: Comparisons of runtime of all recurrent models on the TRAIN-ALL dataset with $d = 800$.

Figure 3 shows the runtime comparison for recurrent models on the TRAIN-ALL dataset. We observe that CTRN is a very scalable and efficient model. Notably, our CTRN model benefits from the training speed brought from the QRNN model, which is clearly significantly faster than LSTM models. Moreover, we also show that CTRN does not significantly increase the runtime of the base QRNN, only incurring an additional $\approx 10s$ per epoch. Moreover, we achieve 4 times faster runtime compared to vanilla LSTM models and 8 times faster than AP-BiLSTM.

Conclusion

We introduced a novel method for jointly learning to compose QA pairs. This is achieved by aligning temporal gates. We show that our lightweight temporal crossing (LTC) mechanism is an effective method of modeling interactions between QA pairs without incurring any parameter cost. Our CTRN model performs competitively on two CQA benchmarks and one factoid QA benchmark while being much faster than LSTM and AP-BiLSTM models.

Acknowledgements

The authors thank anonymous reviewers for their hardwork and feedback.

References

Barrón-Cedeño, A.; Martino, G. D. S.; Joty, S. R.; Moschitti, A.; Al-Obaidli, F.; Romeo, S.; Tymoshenko, K.; and Uva, A. 2016. Convkn at semeval-2016 task 3: Answer and question selection for question answering on arabic and english fora. In *Proceedings of the 10th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2016, San Diego, CA, USA, June 16-17, 2016*.

Bradbury, J.; Merity, S.; Xiong, C.; and Socher, R. 2016. Quasi-recurrent neural networks. *CoRR* abs/1611.01576.

Filice, S.; Croce, D.; Moschitti, A.; and Basili, R. 2016. Kelp at semeval-2016 task 3: Learning semantic relations between questions and answers. In *Proceedings of the 10th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2016, San Diego, CA, USA, June 16-17, 2016*.

He, H.; Gimpel, K.; and Lin, J. J. 2015. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*.

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

Hu, B.; Lu, Z.; Li, H.; and Chen, Q. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Li, J.; Chen, X.; Hovy, E.; and Jurafsky, D. 2015. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*.

Qiu, X., and Huang, X. 2015. Convolutional neural tensor network architecture for community-based question answering. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*.

Rao, J.; He, H.; and Lin, J. J. 2016. Noise-contrastive estimation for answer selection with deep neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*.

Santos; Tan, M.; Xiang, B.; and Zhou, B. 2016. Attentive pooling networks. *CoRR* abs/1602.03609.

Severyn, A., and Moschitti, A. 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, Santiago, Chile, August 9-13, 2015*.

Severyn, A.; Moschitti, A.; Tsagkias, M.; Berendsen, R.; and de Rijke, M. 2014. A syntax-aware re-ranker for microblog retrieval. In *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast , QLD, Australia - July 06 - 11, 2014*.

Tay, Y.; Phan, M. C.; Luu, A. T.; and Hui, S. C. 2017. Learning to rank question answer pairs with holographic dual lstm architecture. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo August 7-11, 2017*.

Tay, Y.; Luu, A. T.; and Hui, S. C. 2017. Enabling efficient question answer retrieval via hyperbolic neural networks. *CoRR* abs/1707.07847.

Wang, M., and Manning, C. D. 2010. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China*.

Wang, D., and Nyberg, E. 2015. A long short-term memory model for answer sentence selection in question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*.

Wang, K.; Ming, Z.; and Chua, T. 2009. A syntactic tree matching approach to finding similar questions in community-based qa services. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*.

Yang, L.; Ai, Q.; Guo, J.; and Croft, W. B. 2016. anmm: Ranking short answer texts with attention-based neural matching model. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*.

Yu, L.; Hermann, K. M.; Blunsom, P.; and Pulman, S. 2014. Deep learning for answer sentence selection. *CoRR* abs/1412.1632.

Zhang, X.; Li, S.; Sha, L.; and Wang, H. 2017. Attentive interactive neural networks for answer selection in community question answering. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*.

Zhou, G.; Cai, L.; Zhao, J.; and Liu, K. 2011. Phrase-based translation model for question retrieval in community question answer archives. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*.