BI-DIRECTIONAL ATTENTION FLOW FOR MACHINE COMPREHENSION

초록
BIDAF(Bi-Directional Attention Flow) 네트워크는 다양한 수준의 세분화에서 컨텍스트를 나타내고 양방향 주의 흐름 메커니즘을 사용하여 조기 요약 없이 쿼리 인식 컨텍스트 표현을 얻는 다단계 계층적 프로세스이다.

1. 서론

Attention model 특징
- 첫째, 계산된 주의 가중치는 문맥을 고정 크기 벡터로 요약하여 질문에 대답하기 위한 문맥에서 가장 관련성이 높은 정보를 추출하는 데 종종 사용된다.
- 둘째, 텍스트 영역에서, 그것들은 종종 시간적으로 동적이며, 현재 시간 단계의 주의 가중치는 이전 시간 단계의 참석 벡터의 함수이다.
- 셋째, 쿼리는 일반적으로 단방향이며, 여기서 쿼리는 컨텍스트 단락 또는 이미지에 참여한다.

(BIDAF) 네트워크
다양한 수준의 세분화 수준에서 컨텍스트 단락의 표현을 모델링하기 위한 계층적 다단계 아키텍처인 양방향 attention flow

BIDAF는 문자 수준, 단어 수준 및 컨텍스트 임베딩을 포함하며 양방향 attention flow를 사용해 query aware context representation을 얻는다.

기존 어텐션 모델의 개선점
: 첫째, BIDAF의 attention layer는 고정 크기 벡터로 컨텍스트 단락을 요약하는 데 사용되지 않는다. 대신, attention은 모든 시간 단계에 대해 계산되고, 각 시간 단계의 attended vector는 이전 layer의 표현과 함께 후속 모델링 layer로 흐를 수 있다.
=> 조기 요약으로 인한 정보 손실을 줄일 수 있습니다.

둘째, BIDAF는 기억력이 없는 attention 메커니즘을 사용한다. 즉, 우리는 Bahdanau et al.(2015)에서와 같이 시간을 통해 attention를 반복적으로 계산하지만, 각 시간 단계의 attention는 현재 시간 단계에서 쿼리와 상황 단락의 함수일 뿐 이전 시간 단계의 주의에 직접적으로 의존하지 않는다. 이러한 단순화가 attention 계층과 모델링 계층 사이의 분업으로 이어진다고 가정한다. attention 계층이 쿼리와 컨텍스트 사이의 attention 학습에 집중하도록 강제하고, 모델링 계층이 쿼리 인식 컨텍스트 표현(attention 계층의 출력) 내에서 상호 작용을 학습하는 데 집중할 수 있도록 한다. 또한 각 시간 단계의 주의는 이전 시간 단계의 잘못된 참석으로 인한 영향을 받지 않습니다. 이 실험이 기억력이 없는 attention이 동적 attention보다 분명한 이점을 제공한다는 것을 보여준다.

셋째, 우리는 서로에게 보완적인 정보를 제공하는 쿼리-대-컨텍스트 및 컨텍스트-대-쿼리 양방향으로 attention 메커니즘을 사용한다.

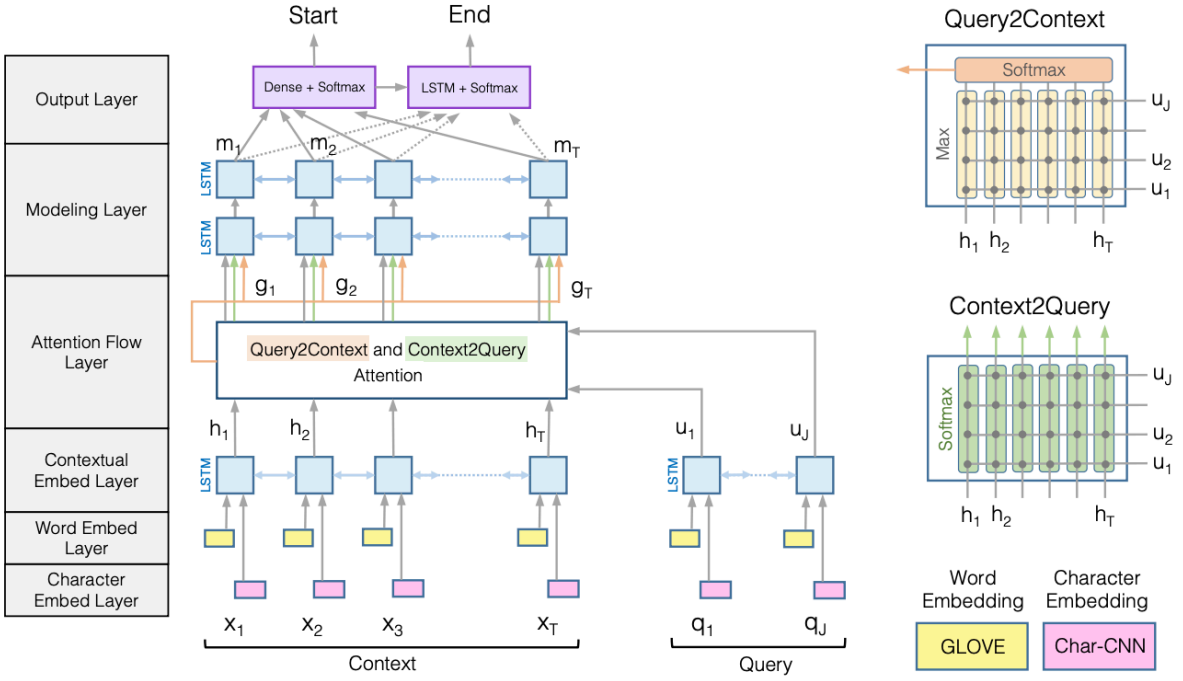

Figure 1: BiDirectional Attention Flow Model *(best viewed in color)*

2. 모델
계층적 다단계 프로세스이며 6개의 계층으로 구성된다.
(그림 1):
1. **Character embedding layer**는 문자 수준 CNN을 사용하여 각 단어를 벡터 공간에 매핑한다.
2. **Word embedding layer**는 사전 훈련된 단어 임베딩 모델을 사용하여 각 단어를 벡터 공간에 매핑한다.
3. **Contextual embedding layer**는 주변 단어의 상황별 신호를 활용하여 단어의 임베딩을 개선한다. 이러한 처음 세 계층은 쿼리와 컨텍스트 모두에 적용된다.
4. **Attention flow layer**은 쿼리와 컨텍스트 벡터를 결합하고 문맥안의 각 단어의 쿼리를 인식하는 피쳐 벡터의 집합을 만든다.
5. **Modeling layer**은 컨텍스트를 스캔하기 위해 순환 신경망을 사용한다.
6. **Output layer**은 쿼리에 대한 답변을 제공합니다.

1.Character embedding layer
문자 임베딩 레이어는 각 단어를 고차원 벡터 공간에 매핑하는 역할을 한다. {x1, .xT} 및 {q1, .qJ}이(가) 각각 입력 컨텍스트 단락 및 쿼리의 단어를 나타내도록 합니다. CNN을 사용하여 각 단어의 문자 수준 임베딩을 얻는다. 문자는 벡터에 내장되며, 이는 CNN에 대한 1D 입력으로 간주될 수 있으며, 크기는 CNN의 입력 채널 크기이다. CNN의 출력은 각 단어에 대한 고정 크기 벡터를 얻기 위해 전체 폭에 걸쳐 최대 풀링된다.

2.Word Embedding Layer
단어 임베딩 레이어는 또한 각 단어를 고차원 벡터 공간에 매핑한다. 사전 훈련된 단어 벡터인 GloVe를 사용하여 각 단어의 고정 단어 임베딩을 얻는다. 문자와 단어 내장 벡터의 연결은 two-layer Highway Network로 전달된다. Highway network의 출력은 2차원의 벡터, 또는 더 편리하게 2개의 행렬이다

$\bar{\mathbf{X}} \in \bar{\mathbb{R}}^{d \times T}$ for the context and $\bar{\mathbf{Q}} \in \mathbb{R}^{d \times J}$ for the query.

3. Contextual Embedding Layer
단어 사이의 시간적 상호 작용을 모델링하기 위해 이전 계층에서 제공하는 임베딩 위에 LSTM을 사용한다. 우리는 LSTM을 양방향으로 배치하고 두 LSTM의 출력을 연결한다.

Hence we obtain $\hat{\mathbf{H}} \in \mathbb{R}^{2d \times T}$ from the context word vectors $\mathbf{X}$, and $\mathbf{U} \in \bar{\mathbb{R}}^{2d \times J}$ from query word vectors $\mathbf{Q}$.

H와 U의 각 열 벡터는 정방향 LSTM과 역방향 LSTM의 출력이 각각 d차원 출력으로 연결되기 때문에 2d차원이다.
모델의 처음 세 계층은 CNN의 다단계 특징 계산과 유사하게 서로 다른 수준의 세분성에서 쿼리와 컨텍스트의 특징을 계산한다는 점에 주목할 필요가 있다.

4. Attention Flow Layer.
Attention 흐름 계층은 정보를 연결하고 융합하는 역할을 합니다. 문맥과 질의어로부터. 이전에 인기 있었던 어텐션 메커니즘과 달리 어텐션 흐름 레이어는 쿼리와 컨텍스트를 단일 특징 벡터로 요약하는 데 사용되지 않는다. 대신에, 어텐션 벡터는 각 시간 단계는 이전 레이어의 임베딩과 함께 후속 모델링 레이어로 흐를 수 있습니다. 이를 통해 조기 요약으로 인한 정보 손실을 줄일 수 있습니다. 레이어에 대한 입력은 컨텍스트 H와 쿼리 U의 컨텍스트 벡터 표현이다.
레이어의 출력은 이전 레이어의 컨텍스트 임베딩과 함께 컨텍스트 단어 G의 쿼리 인식 벡터 표현이다. 이 계층에서, 우리는 문맥에서 쿼리로뿐만 아니라 쿼리에서 문맥으로 어텐션를 계산한다.
아래에서 논의될 이 두 가지 어텐션은 공유된 유사성에서 파생된다.

matrix, $\mathbf{S} \in \mathbb{R}^{T \times J}$, between the contextual embeddings of the context ($\mathbf{H}$) and the query ($\mathbf{U}$), where $\mathbf{S}_{tj}$ indicates the similarity between t-th context word and j-th query word. The similarity matrix is computed by

$$\mathbf{S}_{tj} = \alpha(\mathbf{H}_{:t}, \mathbf{U}_{:j}) \in \mathbb{R} \quad (1)$$

where α is a trainable scalar function that encodes the similarity between its two input vectors, $\mathbf{H}_{:t}$ is t-th column vector of $\mathbf{H}$, and $\mathbf{U}_{:j}$ is j-th column vector of $\mathbf{U}$, We choose $\alpha(\mathbf{h}, \mathbf{u}) = \mathbf{w}_{(\mathbf{S})}^{\top}[\mathbf{h}; \mathbf{u}; \mathbf{h} \circ \mathbf{u}]$, where $\mathbf{w}_{(\mathbf{S})} \in \mathbb{R}^{6d}$ is a trainable weight vector, $\circ$ is elementwise multiplication, $[;]$ is vector concatenation across row, and implicit multiplication is matrix multiplication. Now we use $\mathbf{S}$ to obtain the attentions and the attended vectors in both directions.

H:t H의 t번째 열 벡터
U:j U의 j번째 열벡터

Context-to-query Attention. Context-to-query (C2Q) attention signifies which query words are most relevant to each context word. Let $\mathbf{a}_t \in \mathbb{R}^J$ represent the attention weights on the query words by t-th context word, $\sum \mathbf{a}_{tj} = 1$ for all t. The attention weight is computed by $\mathbf{a}_t = \mathrm{softmax}(\mathbf{S}_{t:}) \in \mathbb{R}^J$, and subsequently each attended query vector is $\tilde{\mathbf{U}}_{:t} = \sum_j \mathbf{a}_{tj}\mathbf{U}_{:j}$. Hence $\tilde{\mathbf{U}}$ is a $2d$-by-T matrix containing the attended query vectors for the entire context.

> C2Q : 어떤 쿼리가 각각의 문맥과 가장 관련 있는지 at : 쿼리에 대한 어텐션 웨이트 U~t : t번째 문맥의 j번째 쿼리의 어텐션 웨이트*쿼리 j번째 쿼리 벡터의 j개수의 합

Query-to-context Attention. Query-to-context (Q2C) attention signifies which context words have the closest similarity to one of the query words and are hence critical for answering the query.

We obtain the attention weights on the context words by $\mathbf{b} = \mathrm{softmax}(\max_{col}(\mathbf{S})) \in \mathbb{R}^T$, where the maximum function ($\max_{col}$) is performed across the column. Then the attended context vector is $\tilde{\mathbf{h}} = \sum_t \mathbf{b}_t \mathbf{H}_{:t} \in \mathbb{R}^{2d}$. This vector indicates the weighted sum of the most important words in the context with respect to the query. $\tilde{\mathbf{h}}$ is tiled T times across the column, thus giving $\tilde{\mathbf{H}} \in \mathbb{R}^{2d \times T}$.

Finally, the contextual embeddings and the attention vectors are combined together to yield $\mathbf{G}$, where each column vector can be considered as the query-aware representation of each context word. We define $\mathbf{G}$ by

$$\mathbf{G}_{:t} = \boldsymbol{\beta}(\mathbf{H}_{:t}, \tilde{\mathbf{U}}_{:t}, \tilde{\mathbf{H}}_{:t}) \in \mathbb{R}^{d_{\mathbf{G}}} \tag{2}$$

where $\mathbf{G}_{:t}$ is the t-th column vector (corresponding to t-th context word), $\boldsymbol{\beta}$ is a trainable vector function that fuses its (three) input vectors, and $d_{\mathbf{G}}$ is the output dimension of the $\boldsymbol{\beta}$ function. While the $\boldsymbol{\beta}$ function can be an arbitrary trainable neural network, such as multi-layer perceptron, a simple concatenation as following still shows good performance in our experiments: $\boldsymbol{\beta}(\mathbf{h}, \tilde{\mathbf{u}}, \tilde{\mathbf{h}}) = [\mathbf{h}; \tilde{\mathbf{u}}; \mathbf{h} \circ \tilde{\mathbf{u}}; \mathbf{h} \circ \tilde{\mathbf{h}}] \in \mathbb{R}^{8d \times T}$ (i.e., $d_{\mathbf{G}} = 8d$).

> Q2C : 어떤 문맥이 어떤 쿼리와 가장 유사하고 쿼리에 대한 응답에 중요한지, b: 어텐션 웨이트(S의 max col)
h~벡터는 쿼리에 대한 문맥에서 가장 중요한 단어의 가중합을 의미

> contextual embedding 과 attention벡터를 G로 합친다. => query aware representation of each context word

5. **Modeling Layer**
모델링 계층에 대한 입력은 G로, 컨텍스트 단어의 쿼리 인식 표현을 인코딩한다. 모델링 계층의 출력은 쿼리에서 조건화된 컨텍스트 단어 간의 상호 작용을 캡처한다. 이는 쿼리와 무관하게 컨텍스트 단어 간의 상호 작용을 캡처하는 컨텍스트 임베딩 계층과 다르다. 우리는 각 방향에 대해 출력 크기가 d인 양방향 LSTM의 두 레이어를 사용한다. 따라서 우리는 행렬을 얻는다.
M (차원 R (2d×T)) - 답을 예측하기 위해 출력 레이어에 전달된다. M의 각 열 벡터는 전체 문맥 단락 및 쿼리에 대한 단어에 대한 문맥 정보를 포함할 것으로 예상된다.

6.output layer

출력 계층은 애플리케이션별로 다르다 여기서는 QA 작업에 대한 출력 계층에 대해 설명한다. QA 작업을 수행하려면 모델이 쿼리에 응답할 문단의 하위 구문을 찾아야 한다. 문구는 문단에서 문구의 시작 및 끝 인덱스를 예측하여 도출된다. 우리는 전체 단락에 대한 시작 지수의 확률 분포를 다음과 같이 구한다.

$$\mathbf{p}^1 = \text{softmax}(\mathbf{w}_{(\mathbf{p}^1)}^\top [\mathbf{G}; \mathbf{M}]), \tag{3}$$

where $\mathbf{w}_{(\mathbf{p}^1)} \in \mathbb{R}^{10d}$ is a trainable weight vector. For the end index of the answer phrase, we pass $\mathbf{M}$ to another bidirectional LSTM layer and obtain $\mathbf{M}^2 \in \mathbb{R}^{2d \times T}$. Then we use $\mathbf{M}^2$ to obtain the probability distribution of the end index in a similar manner:

$$\mathbf{p}^2 = \text{softmax}(\mathbf{w}_{(\mathbf{p}^2)}^\top [\mathbf{G}; \mathbf{M}^2]) \tag{4}$$

Training. We define the training loss (to be minimized) as the sum of the negative log probabilities of the true start and end indices by the predicted distributions, averaged over all examples:

$$L(\theta) = -\frac{1}{N} \sum_i^N \log(\mathbf{p}^1_{y_i^1}) + \log(\mathbf{p}^2_{y_i^2}) \tag{5}$$

where θ is the set of all trainable weights in the model (the weights and biases of CNN filters and LSTM cells, $\mathbf{w}_{(\mathbf{S})}$, $\mathbf{w}_{(\mathbf{p}^1)}$ and $\mathbf{w}_{(\mathbf{p}^2)}$), N is the number of examples in the dataset, y_i^1 and y_i^2 are the true start and end indices of the i-th example, respectively, and $\mathbf{p}_k$ indicates the k-th value of the vector $\mathbf{p}$.

Test. The answer span (k, l) where $k \leq l$ with the maximum value of $\mathbf{p}_k^1 \mathbf{p}_l^2$ is chosen, which can be computed in linear time with dynamic programming.