TVQA: Localized Compositional Video Question Answering

Moonsu Han

MLAI, KAIST

What Are We Going to Learn

The contents of this lecture is as follows:

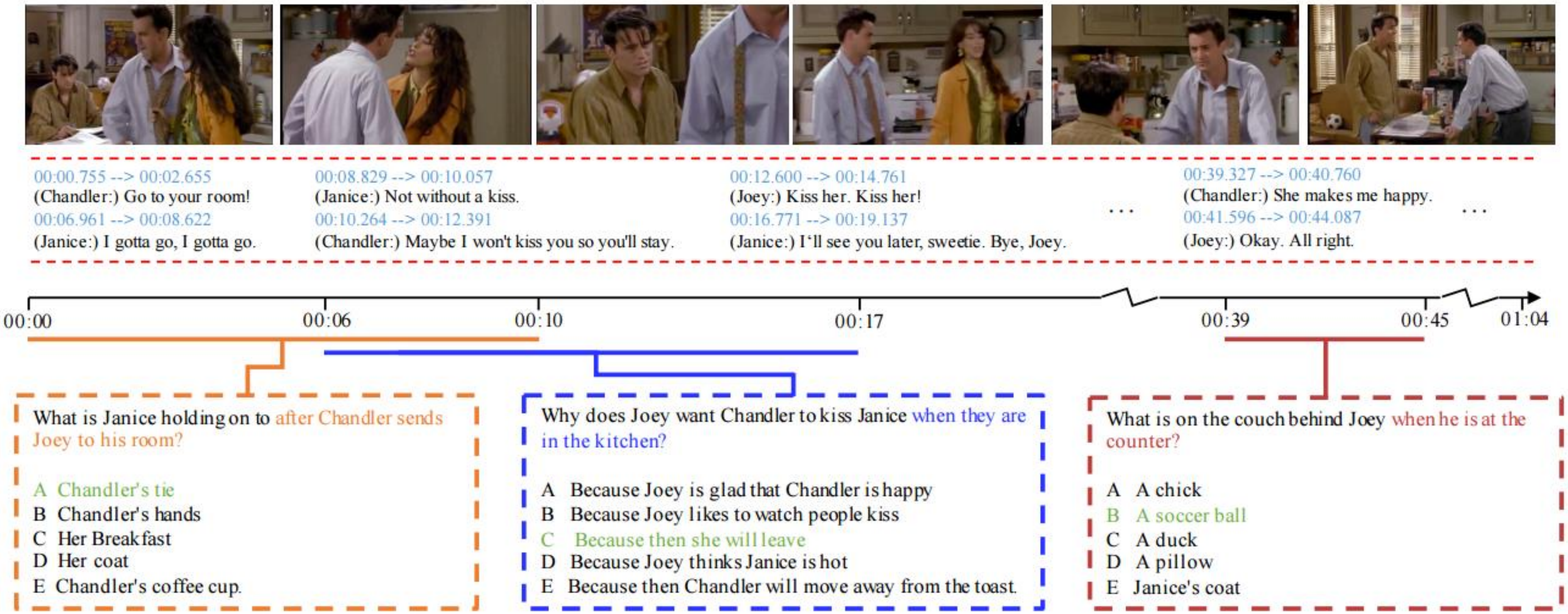
1. TVQA dataset and its characteristic

2. Introduce for Multi-modal Video QA model with its composition and operation

3. Code review for Multi-modal Video QA model

What is the TVQA Dataset?

TVQA [Lei18] is a localized, compositional video question answering dataset containing 153K question-answer pairs from 22K clips in 6 TV series.



[Lei18] J. Lei, L. Yu, M. Bansal, T. L. Berg, TVQA: Localized, Compositional Video Question Answering. EMNLP 2018

Dataset Collection

Amazon Mechanical Turk was used for VQA collection on video clips, where workers were presented with ***both videos and aligned named subtitles***.

00:03 → UNKNAME: Hey. I got some bad news.
(Ellipsis)
01:31 → UNKNAME: Your food is abysmal!

Question & Answers

[Lei18] J. Lei, L. Yu, M. Bansal, T. L. Berg, TVQA: Localized, Compositional Video Question Answering. EMNLP 2018

Difference from Existing Datasets

After extracting question answer pairs based on its subtitle, existing datasets ***added the frames corresponding to each subtitle***.

00:03 → UNKNAME: Hey. I got some bad news.
(Ellipsis)
01:31 → UNKNAME: Your food is abysmal!

Question & Answers

Dataset	V. Src.	QType	#Clips / #QAs	Avg. Len.(s)	Total Len.(h)	Q. Src. text	Q. Src. video	Timestamp annotation
MovieFIB (Maharaj et al., 2017a)	Movie	OE	118.5k / 349k	4.1	135	✓	-	-
Movie-QA (Tapaswi et al., 2016)	Movie	MC	6.8k / 6.5k	202.7	381	✓	-	✓
TGIF-QA (Jang et al., 2017)	Tumblr	OE&MC	71.7k / 165.2k	3.1	61.8	✓	✓	-
Pororo-QA (Kim et al., 2017)	Cartoon	MC	16.1k / 8.9k	1.4	6.3	✓	✓	-
TVQA (our)	TV show	MC	21.8k / 152.5k	76.2	461.2	✓	✓	✓

Comparison between VQA datasets
(OE = open-ended, MC = Multiple-choices, Q. Src. = Question Source)

TVQA tried to solve this limitation by ***collecting question answer pairs from both visual and text information***.

[Lei18] J. Lei, L. Yu, M. Bansal, T. L. Berg, TVQA: Localized, Compositional Video Question Answering. EMNLP 2018

Difference from Visual Question Answering

Visual question answering model ***takes an image and question*** as input and ***outputs exact word for an answer***.

"How many zebras are in this image?" → "2"

Visual Question Answering

Difference from Visual Question Answering

In video question answering task, the model ***takes a question, subtitle, frame and answer*** as input and ***outputs score of the answers***.

[Question]
"What is Kutner writing on when talking to Lawrence?

00:03 → UNKNAME: Hey. I got some bad news.
01:31 → UNKNAME: Your food is abysmal!

[Answer 1] Kutner is writing on a clipboard.
[Answer 2] Kutner is writing on a laptop.
[Answer 3] Kutner is writing on a notepad.
[Answer 4] Kutner is writing on an index card.
[Answer 5] Kutner is writing on his hand.

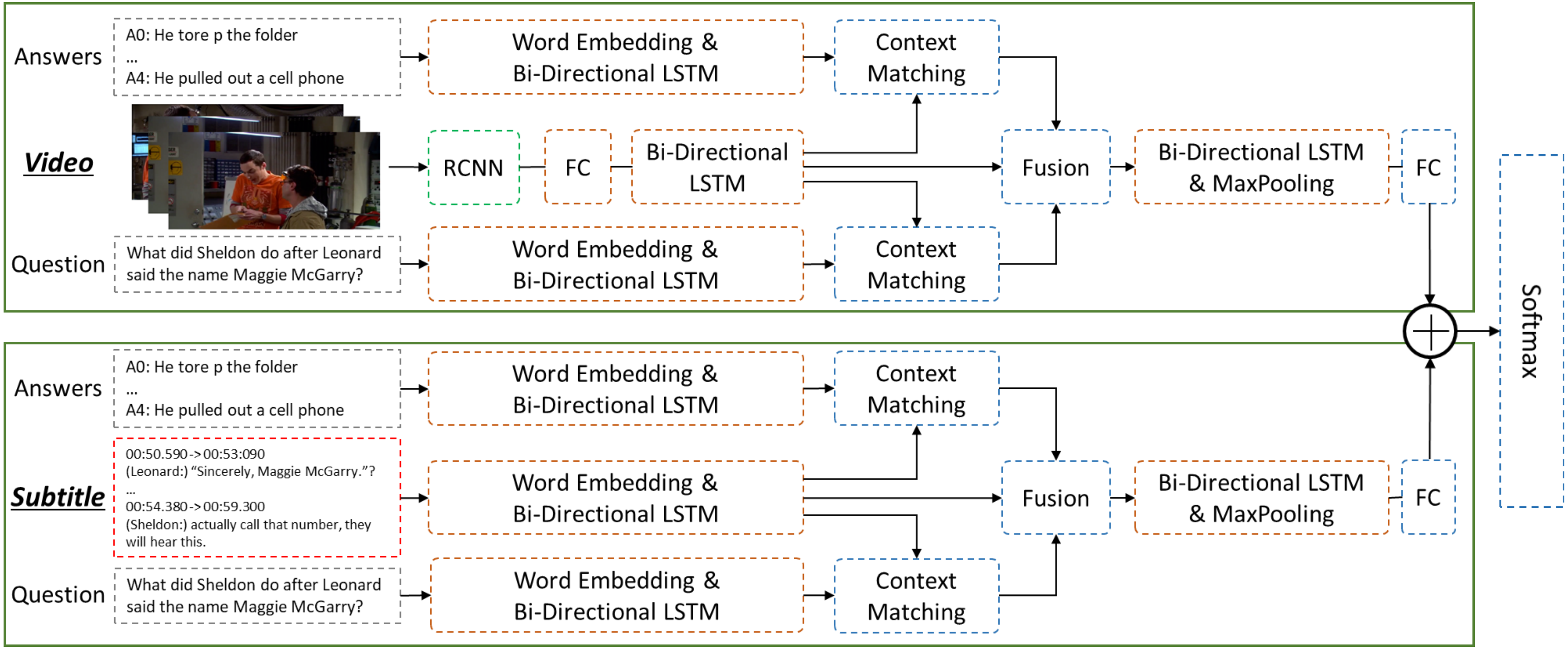
→ Score of the answers

Video Question Answering

VQA task requires a model that is able to understand multi-modal information from spatio-temporal format.

Multi-Modal Video QA

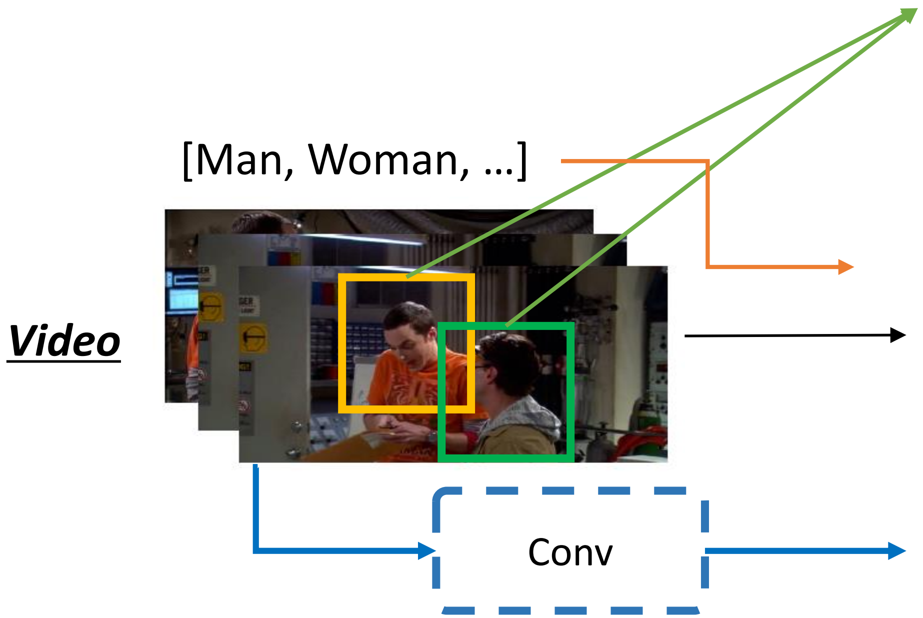
In this lecture, we explore ***Multi-Modal Video QA*** which is first attempt in order to solve TVQA dataset.



[Lei18] J. Lei, L. Yu, M. Bansal, T. L. Berg, TVQA: Localized, Compositional Video Question Answering. EMNLP 2018

Spatial Information Extraction from a Video

It used ***three different type of extracting methods*** in order to capture spatial information.

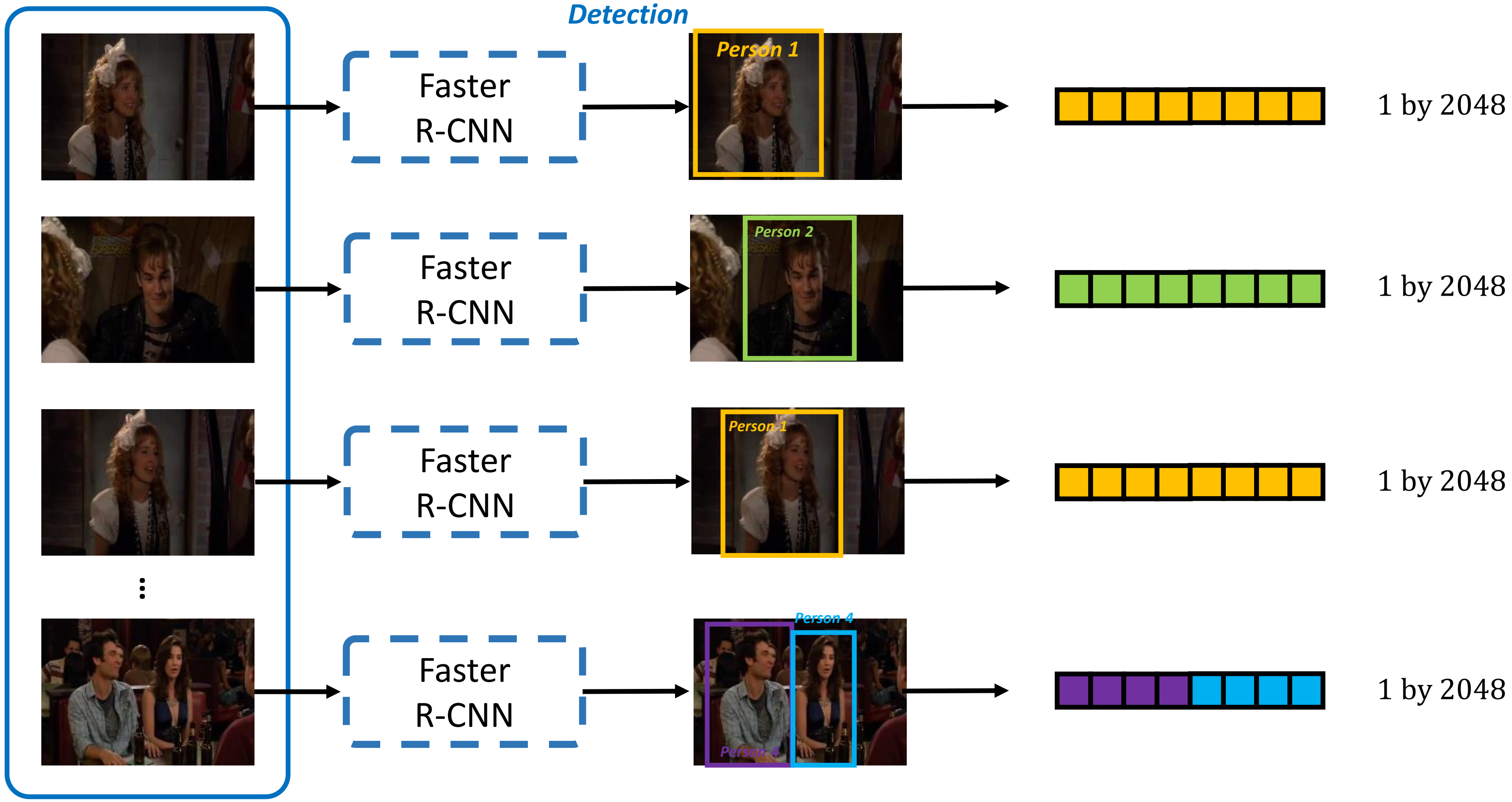


1) Regional Visual Features
   Selecting regions from each detected label
2) Visual Concepts Features
   Using ***detected labels*** as visual inputs
3) ImageNet Features
   Extracting ***last block of ResNet101***

[Lei18] J. Lei, L. Yu, M. Bansal, T. L. Berg, TVQA: Localized, Compositional Video Question Answering. EMNLP 2018

Regional Visual Features

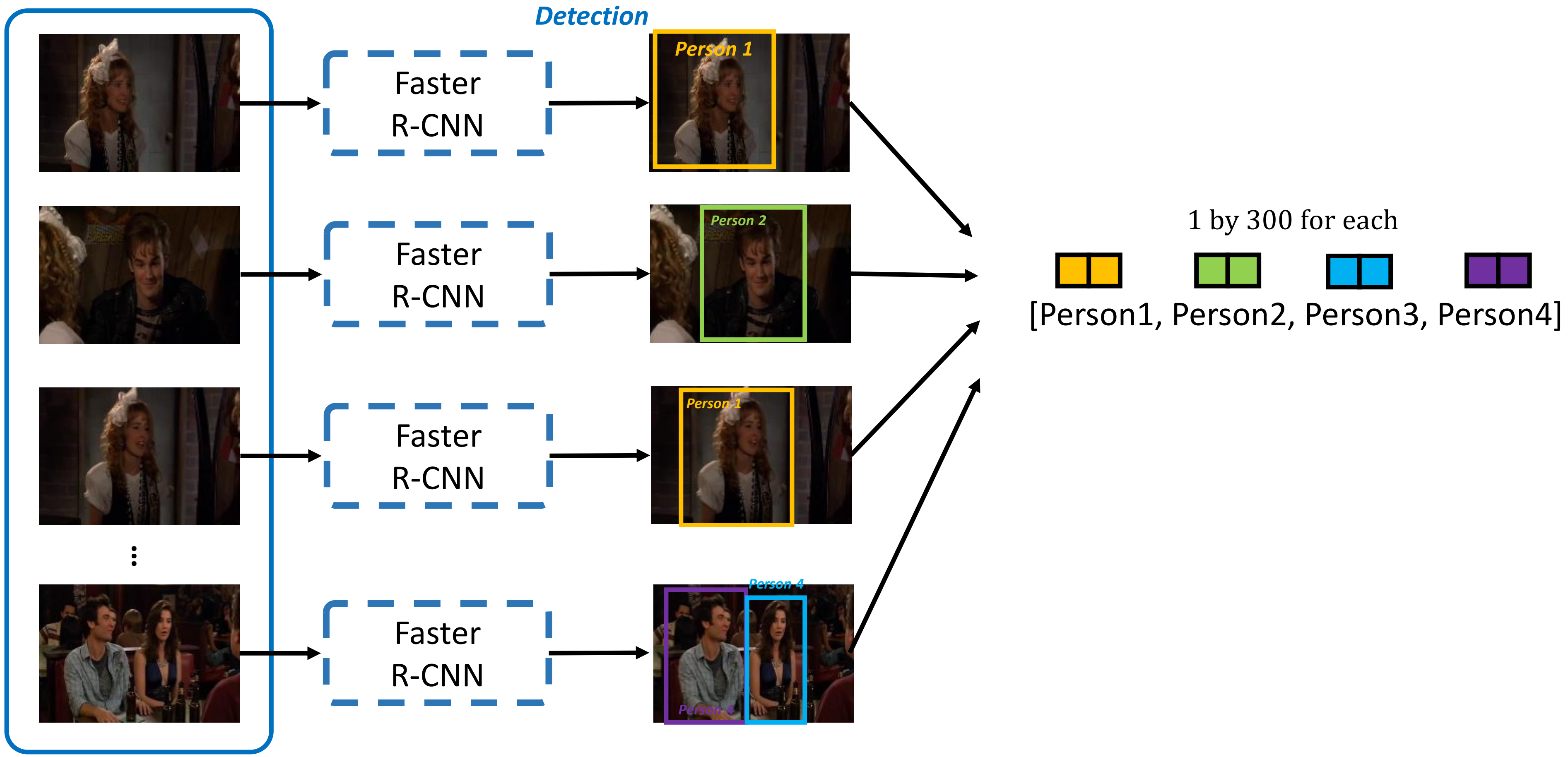
The model ***extracts spatial information*** from a video using ***object detection network*** [Ren15].



[Ren15] S. Ren, K. He, R. B. Girshick, J. Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS 2015

Visual Concepts Features

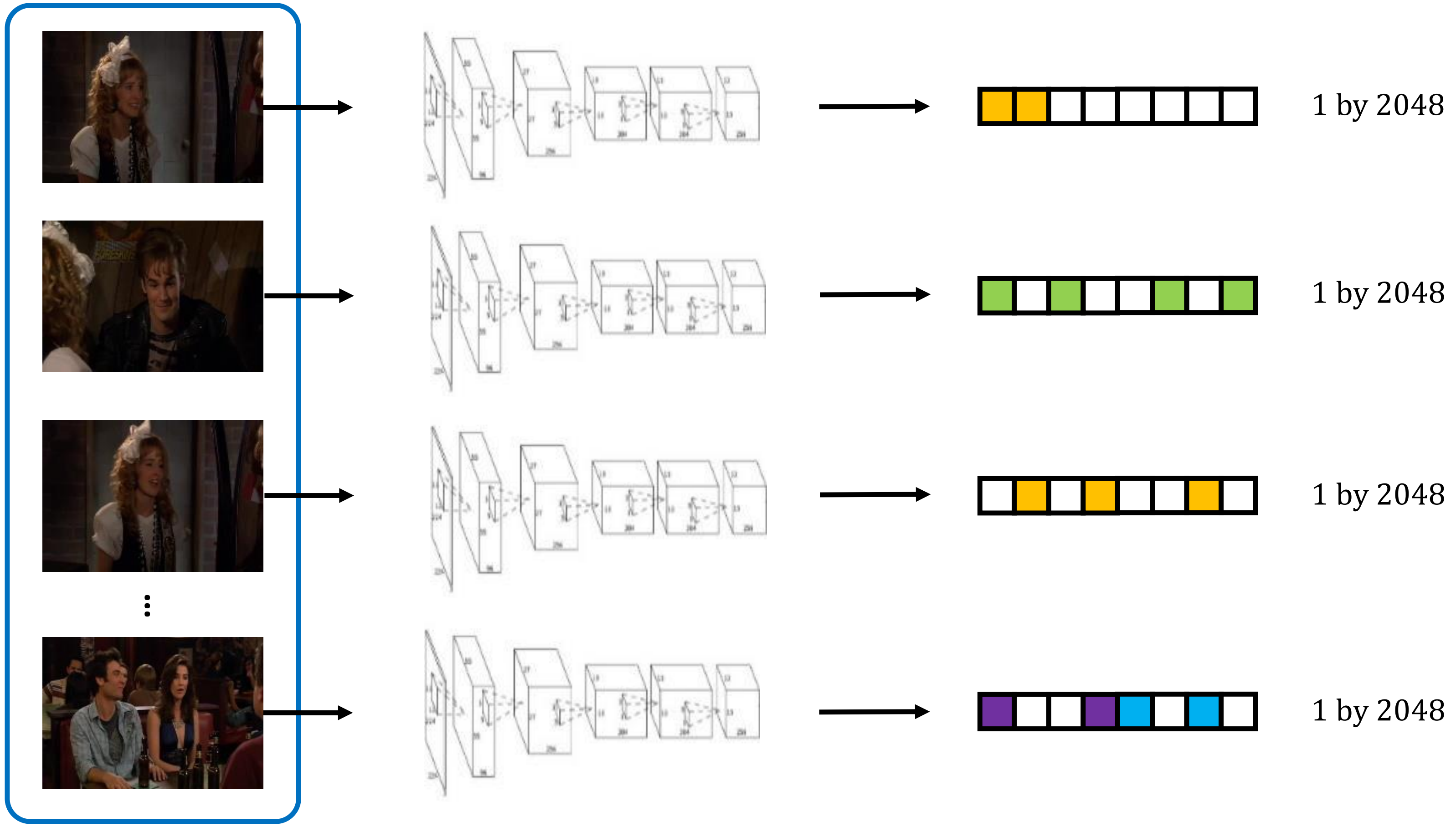
[Yu18] found that ***using detected object labels*** as input to an image ***gave comparable performance*** to using CNN features directly.



[Yu18] L. Yu, Z. Lin, X. Shen, J. Yang, X. Lu, M. Bansal, T. L. Berg, MAttNet: Modular Attention Network for Referring Expression Comprehension. CVPR 2018

ImageNet Features

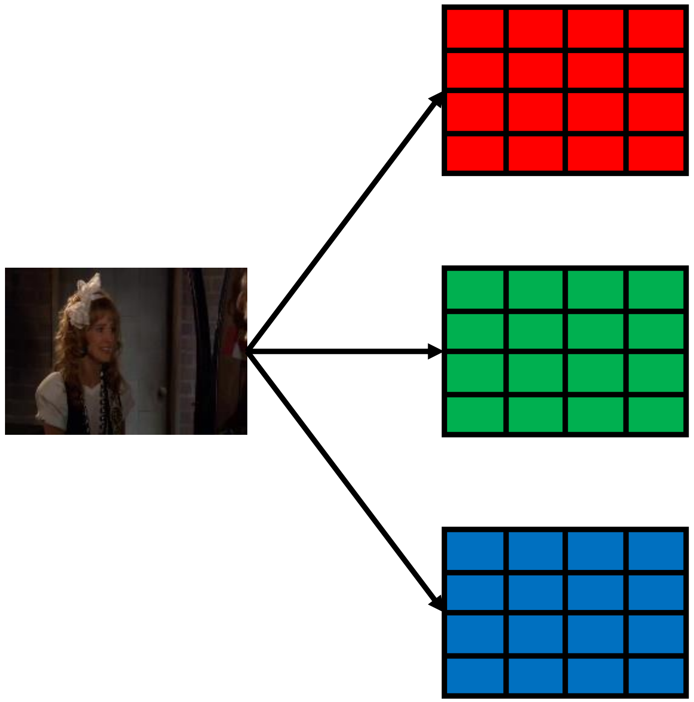
It is the simplest way to ***extract spatial information*** by using ***famous CNN architecture*** such as ResNet101 [He16].

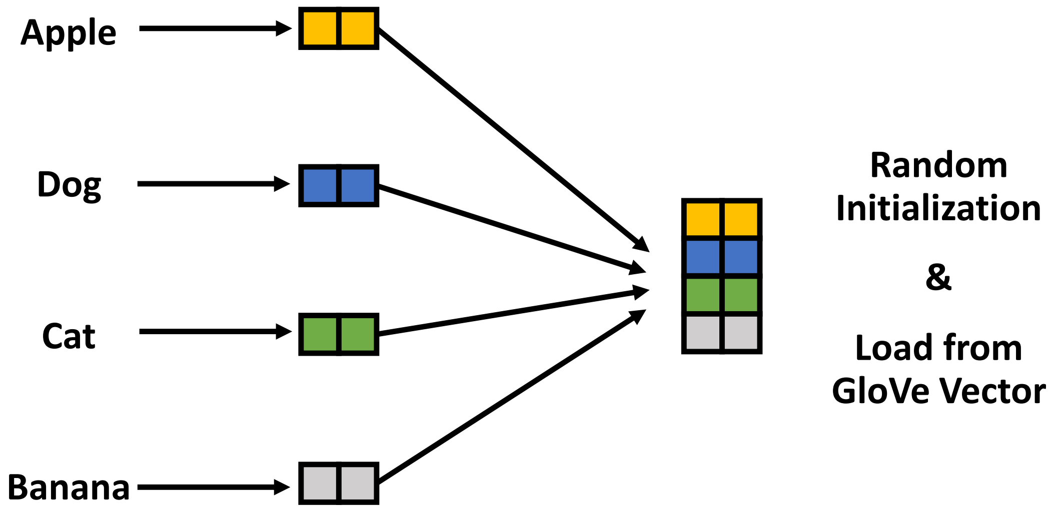


[He16] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition. CVPR 2016

Word Representation

Unlike an image represented as RGB channels, how can we represent a word to other format?

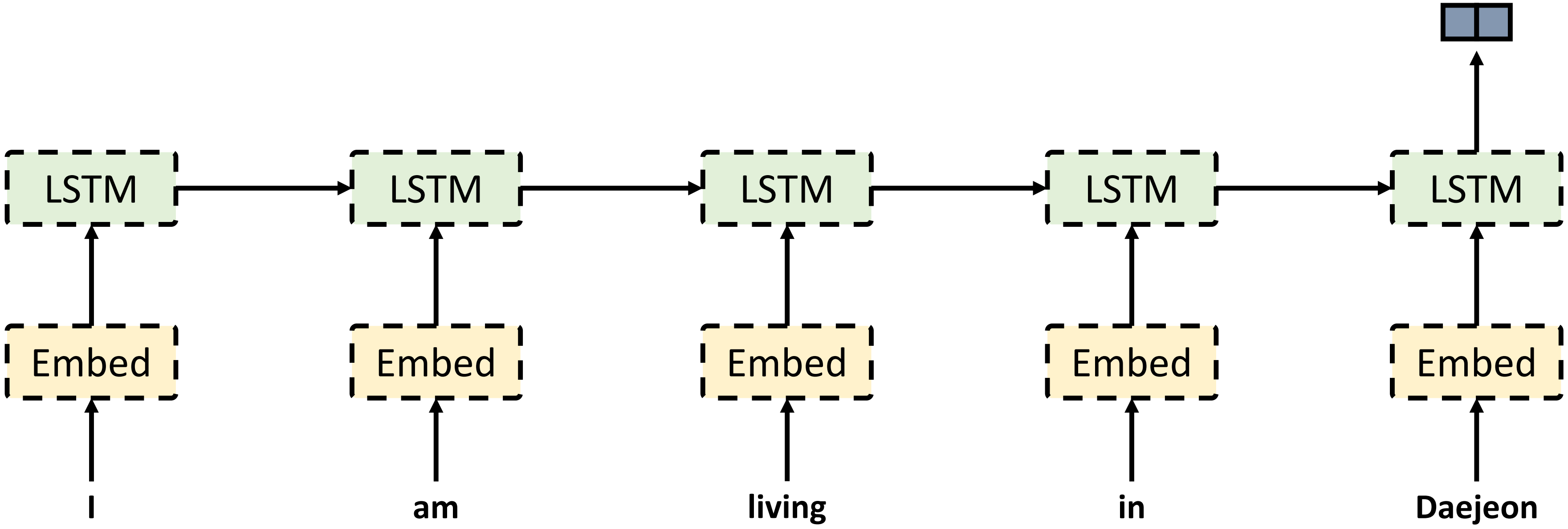




We ***can represent a word as a vector*** and ***synthesize all words to create a single vocabulary***.

Sentence Representation

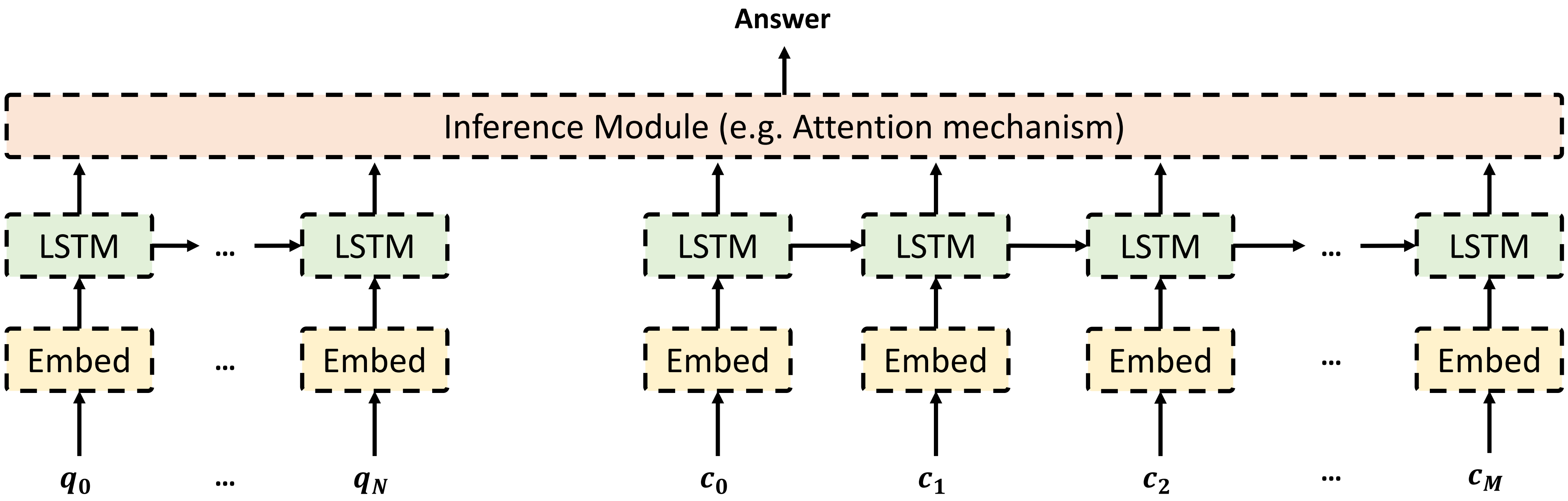
After generating the representation for words, there need a module to encode words to one.



RNN can extract important representation among them.

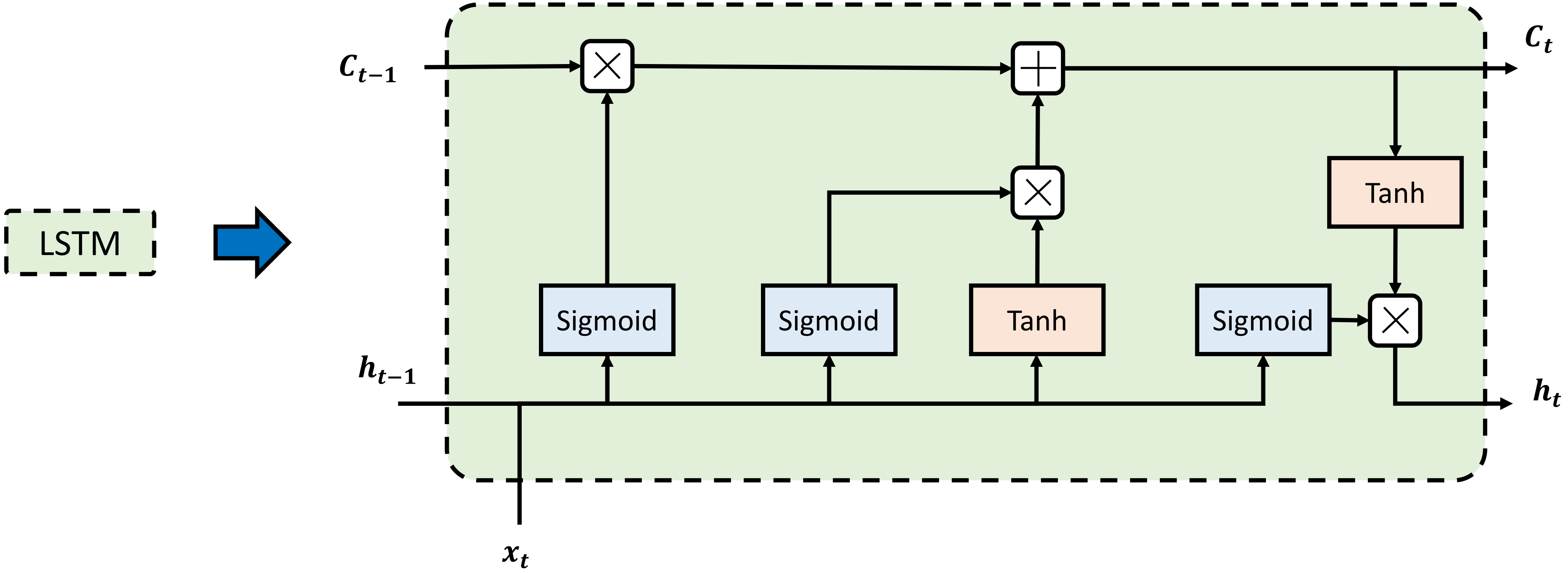
Representation on Question Answering Task

In question answering task, ***the hidden states for each LSTM*** is used for ***a word representation for each word***.



Long Short-Term Network

In general manner, many researchers utilize ***Long Short-Term Network to capture important representation***.

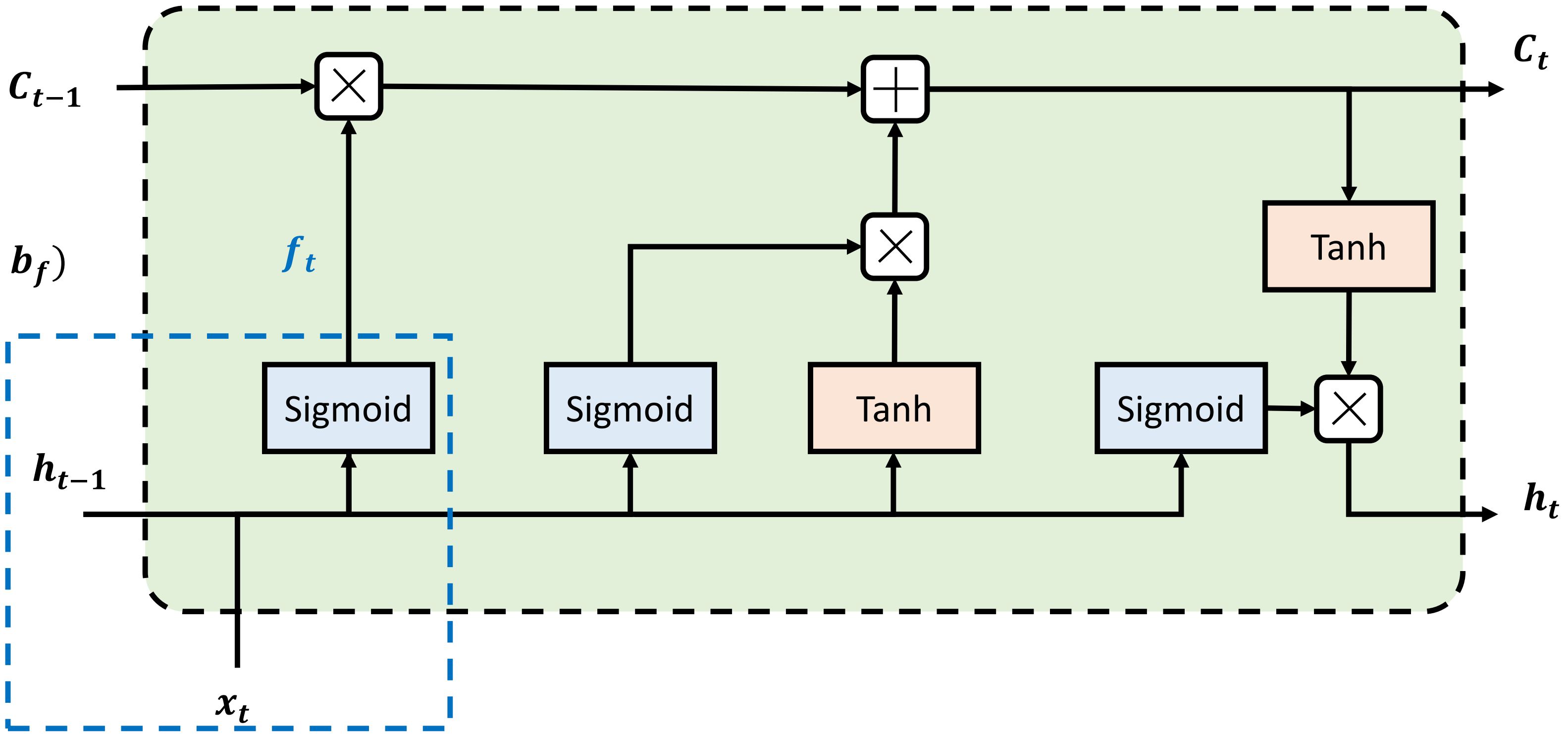


[Hochreiter and Schmidhuber97] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory. Neural Computation 1997

Forget Gate Layer (LSTM)

Forget gate layer decides ***what information should be thrown away or kept***.

$$f_t = sigmoid(W_f \cdot [h_{t-1}, x_t] + b_f)$$



The closer to 0 means to forget, and the closer 1 means to keep.

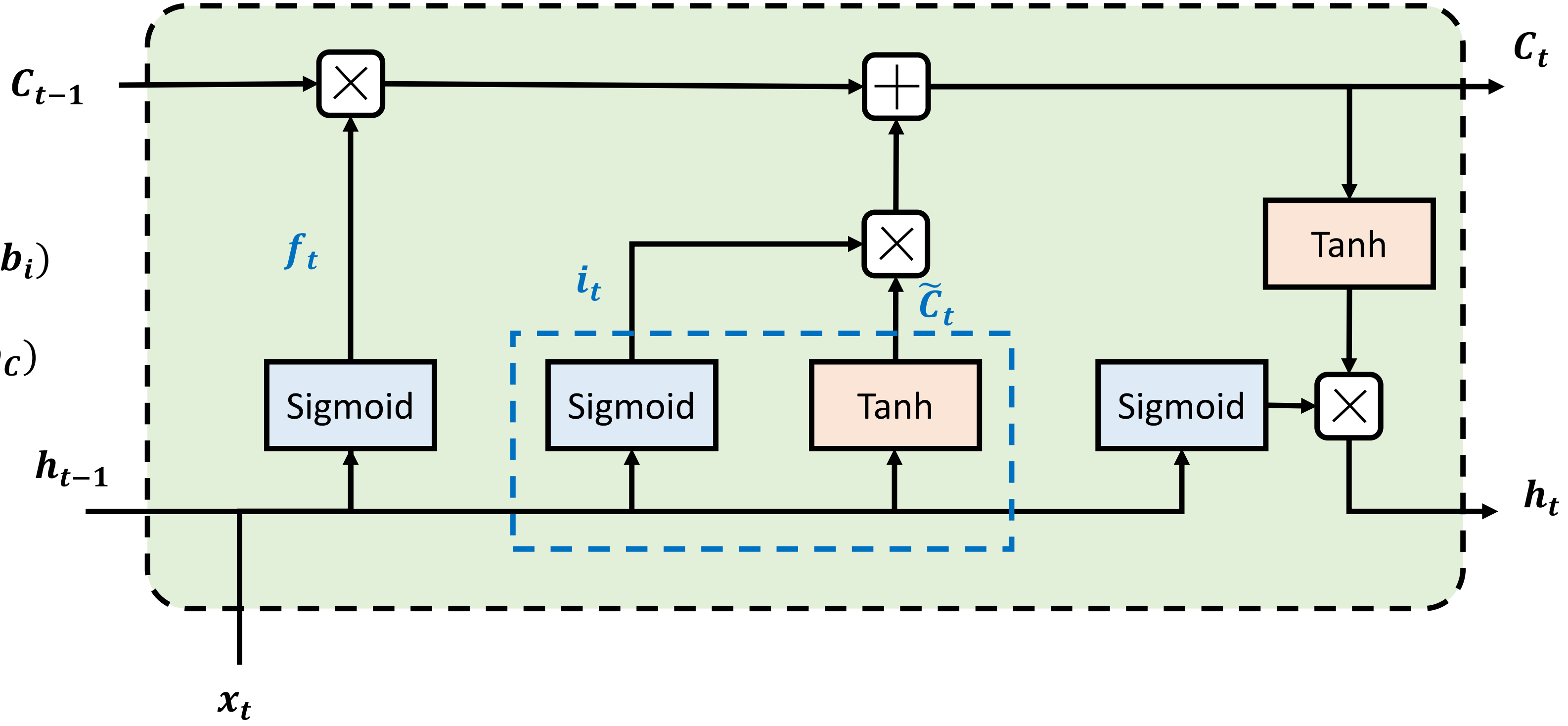
[Hochreiter and Schmidhuber97] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory. Neural Computation 1997

Input Gate Layer (LSTM)

Input gate layer is to update the cell state, which means ***how it takes the information from the current input and previous hidden state***.

$$i_t = sigmoid(i_f \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

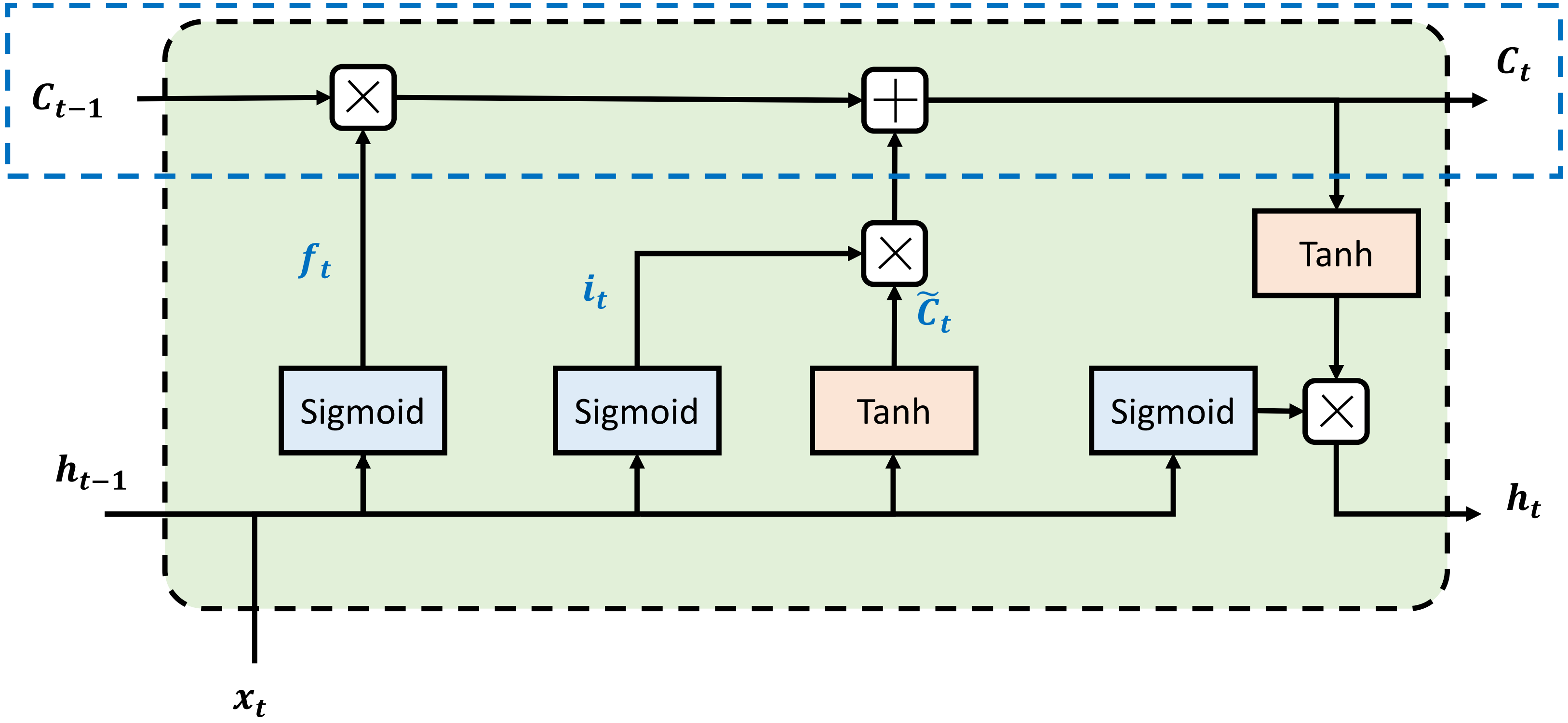


[Hochreiter and Schmidhuber97] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory. Neural Computation 1997

Cell State Update (LSTM)

From last two layers, it ***combines the previous cell state with the information***.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t$$



Namely, it is the process of making representative information.

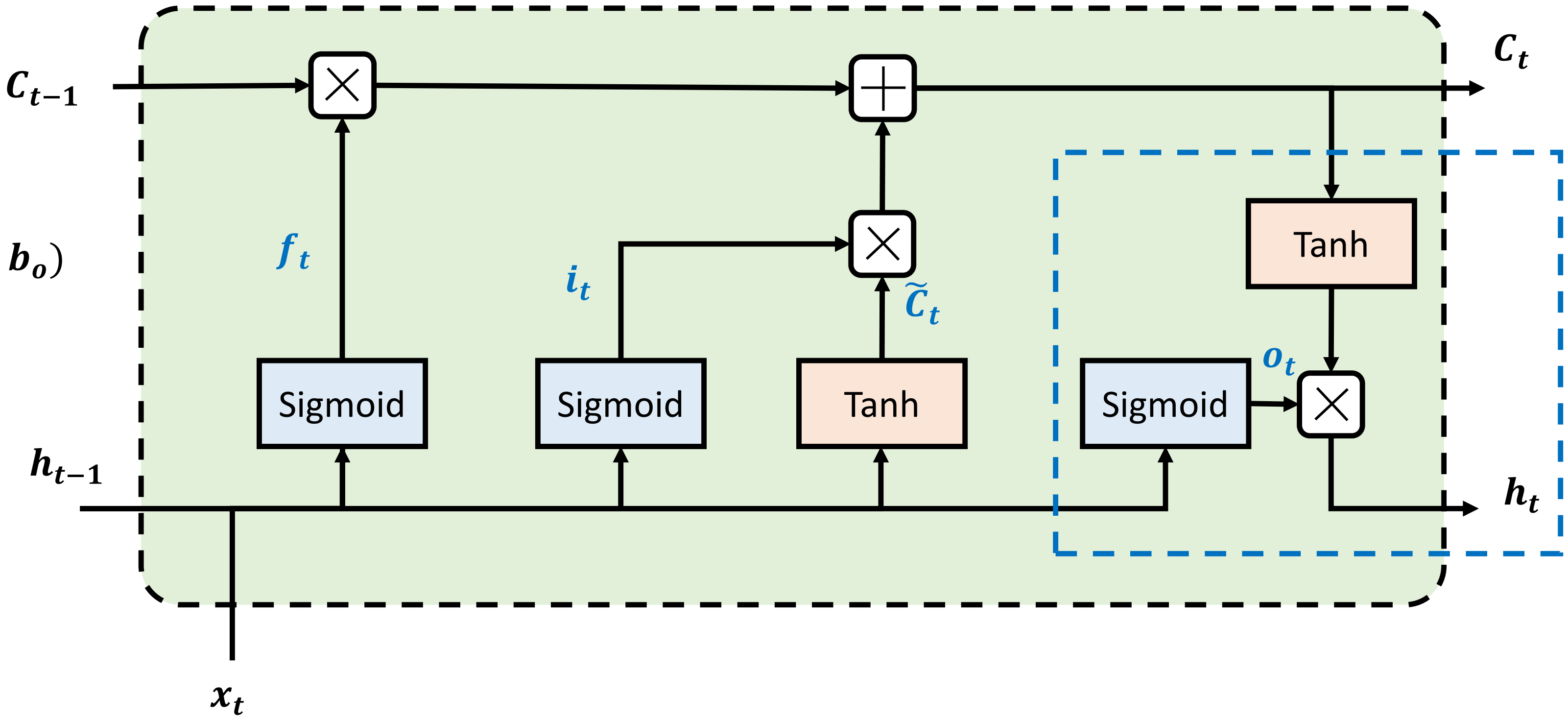
[Hochreiter and Schmidhuber97] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory. Neural Computation 1997

Output Gate Layer (LSTM)

Output gate layer decides ***what the next hidden state should be***.

$$o_t = sigmoid(W_o \cdot [h_{t-1}, x_t] + b_o)$$

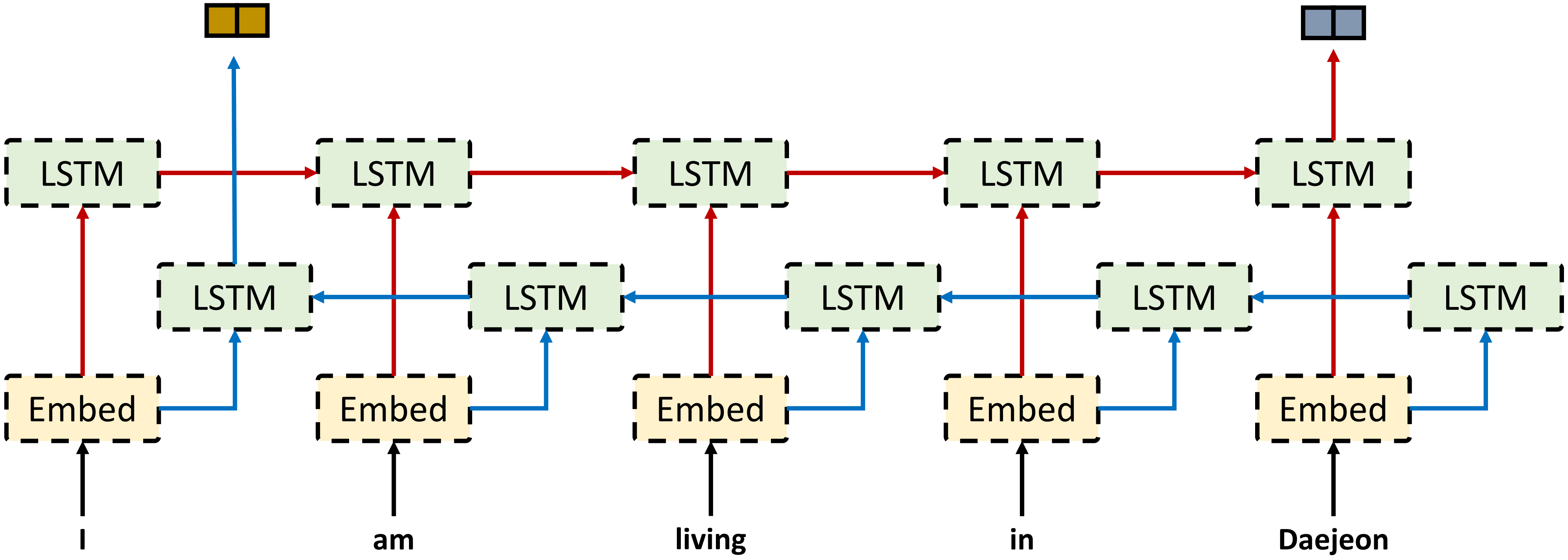
$$h_t = o_t * tanh(C_t)$$



[Hochreiter and Schmidhuber97] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory. Neural Computation 1997

Bi-directional LSTM

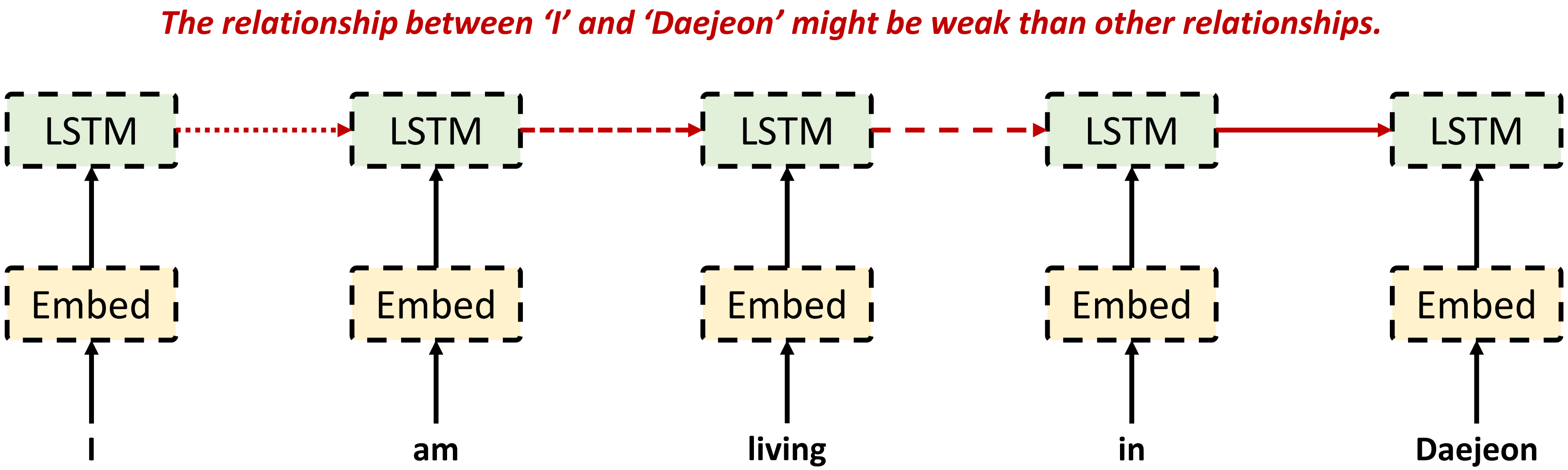
Uni-directional LSTM only consider a single forward pass, therefore, Bi-directional LSTM has been proposed.



[Schuster97] M. Schuster, K. K. Paliwal, Bidirectional recurrent neural networks. IEEE Trans. Signal Processing 1997

Limitation of Long Short-Term Network

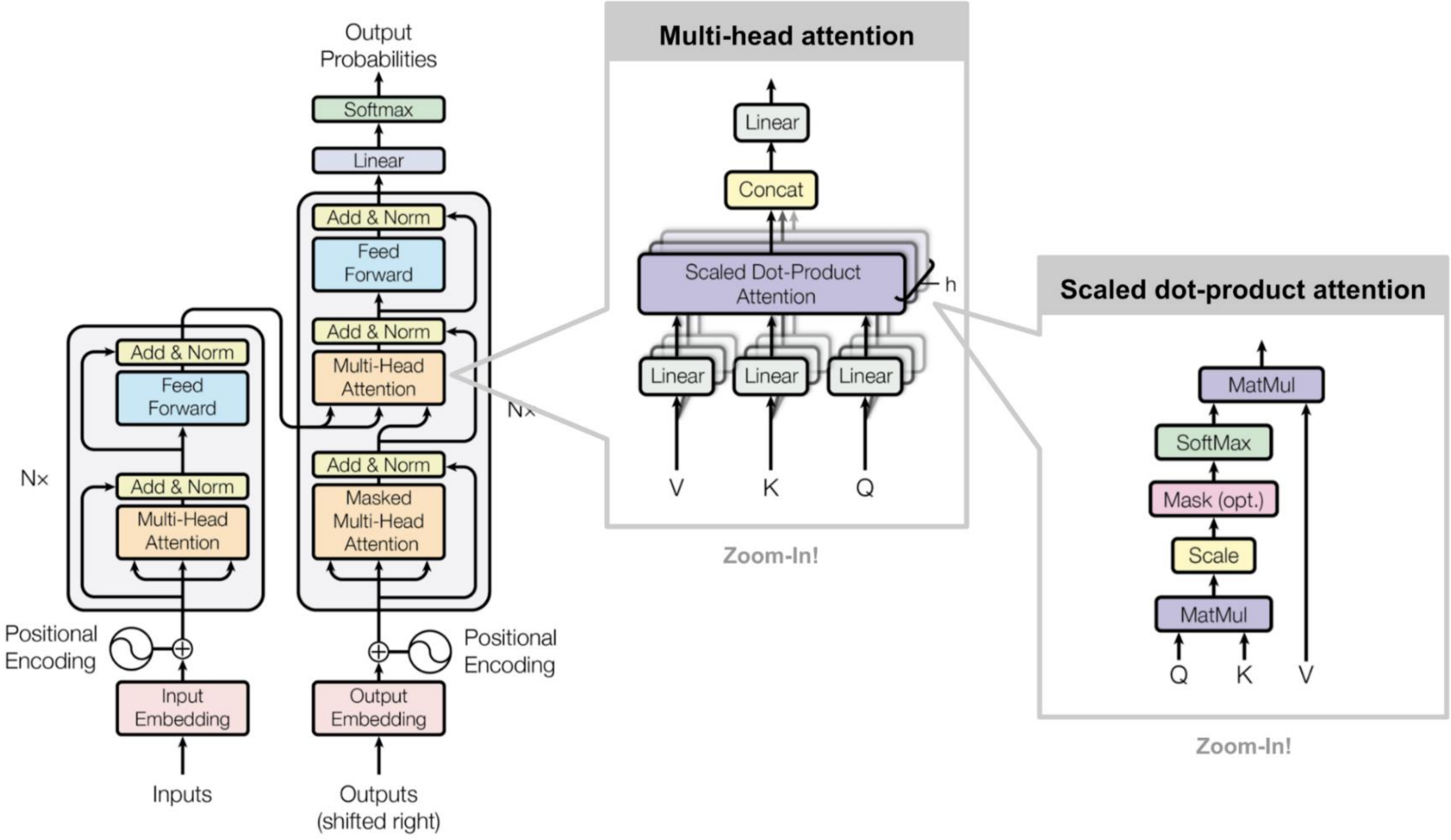
Despite of considering Bi-directional way, it ***cannot compare the representation equally***.



Also, it ***cannot work well when the sequence is long***.

Transformer

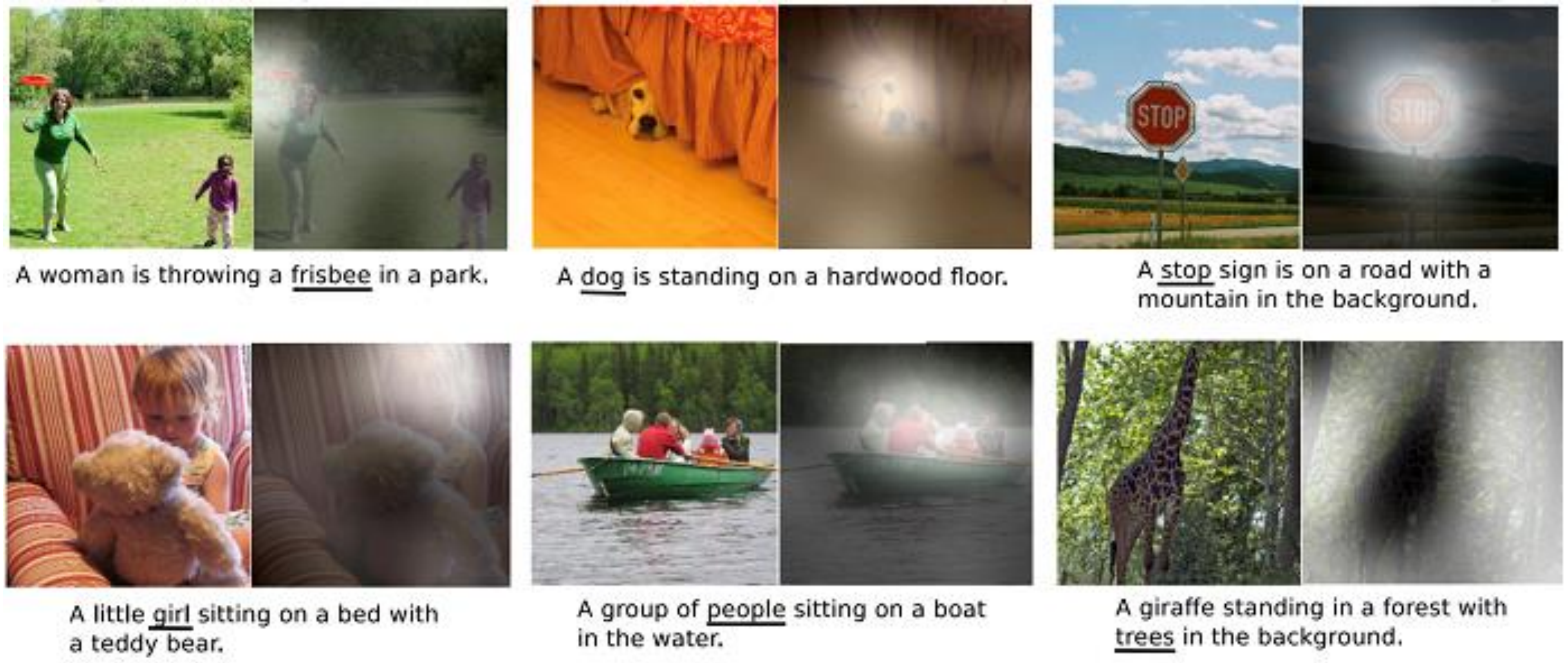
[Vaswani17] proposed ***a self-attention mechanism*** to ***compare the representations equally***.



[Vaswani17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is All you Need. NIPS 2017

Attention Mechanism

After generating the representations, ***attention mechanism can generate a high-level reasoning information***.

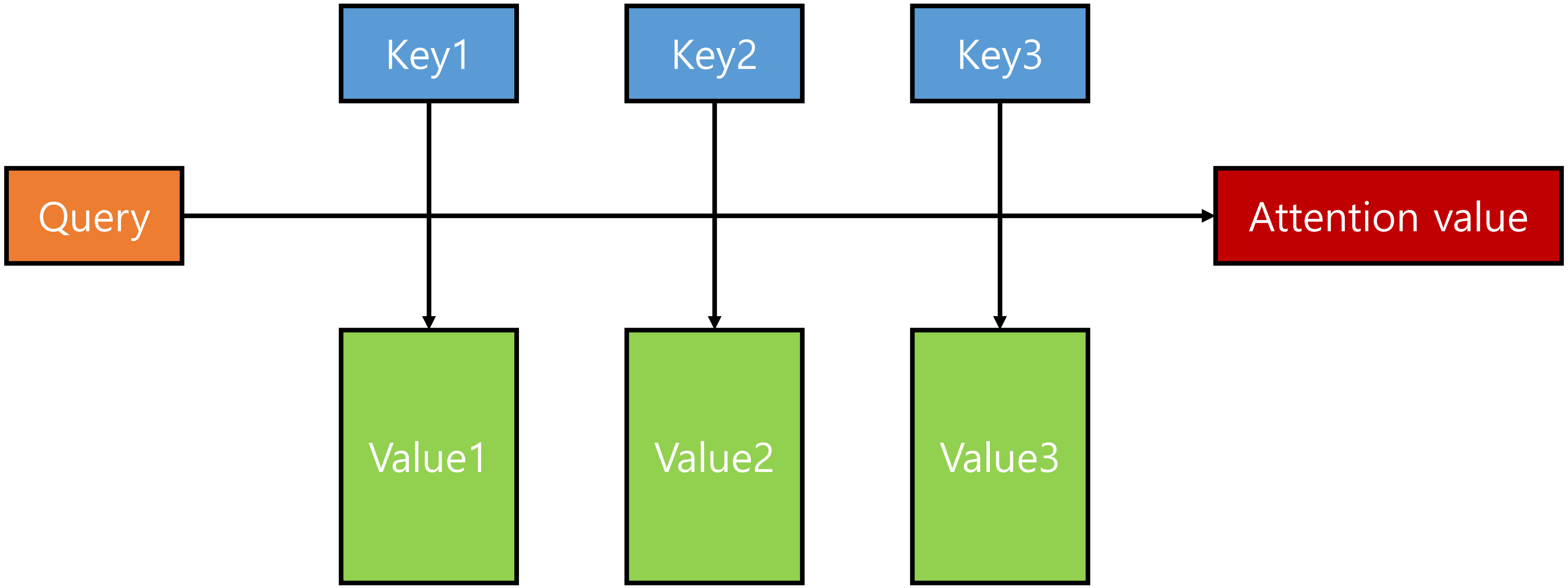


This method is ***interpretable***, which means a person can analyze why a word is outputted from the input.

[Xu15] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, Y. Bengio, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML 2015

Attention Mechanism

Attention value is computed by ***Keys***, ***Query*** and ***Values***.

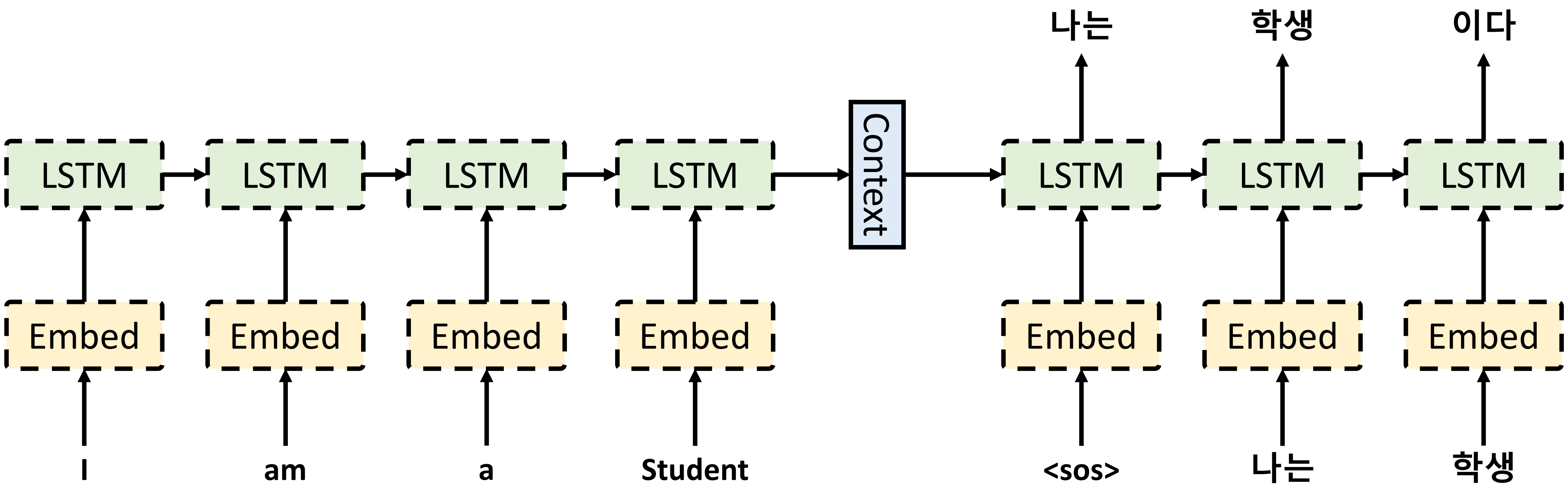


Attention(Q, K, V) = Attention value

https://wikidocs.net/22893

Sequence to Sequence Model (Seq2seq)

Sequence to sequence model is used in machine translation, speech recognition and video captioning.

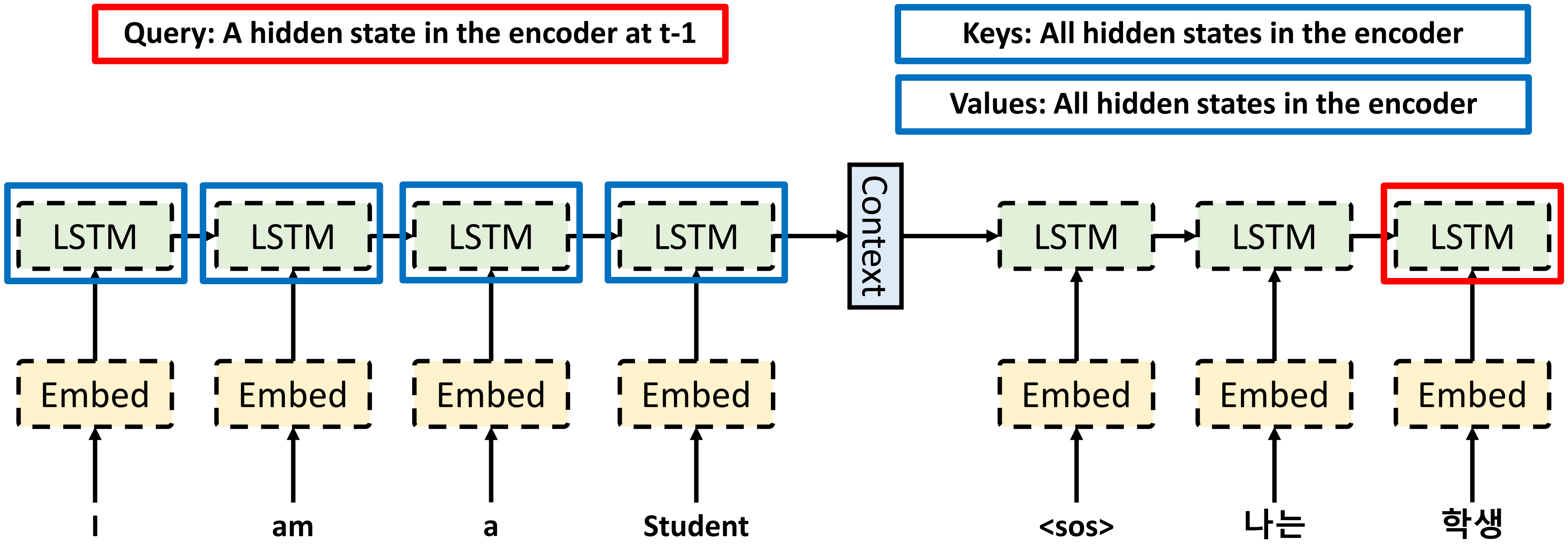


$$s_t = f(s_{t-1}, y_{t-1})$$

https://wikidocs.net/22893

Attention Mechanism on Seq2seq Model

In Seq2seq model, ***three components*** for using attention mechanism ***come from the encoder and decoder***.

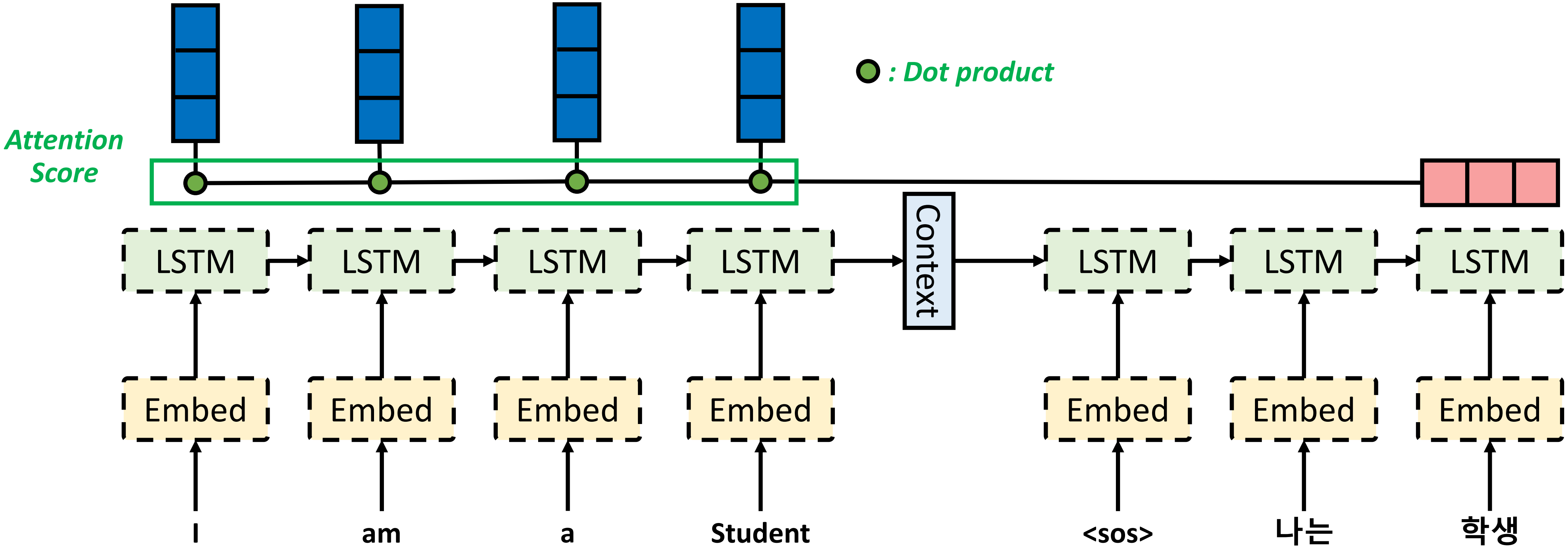


$$s_t = f(s_{t-1}, y_{t-1}, \alpha_t)$$

https://wikidocs.net/22893

Calculation for Attention Score

Attention score is ***a score measuring similarity*** between ***previous hidden state*** and ***all hidden states from the encoder***.



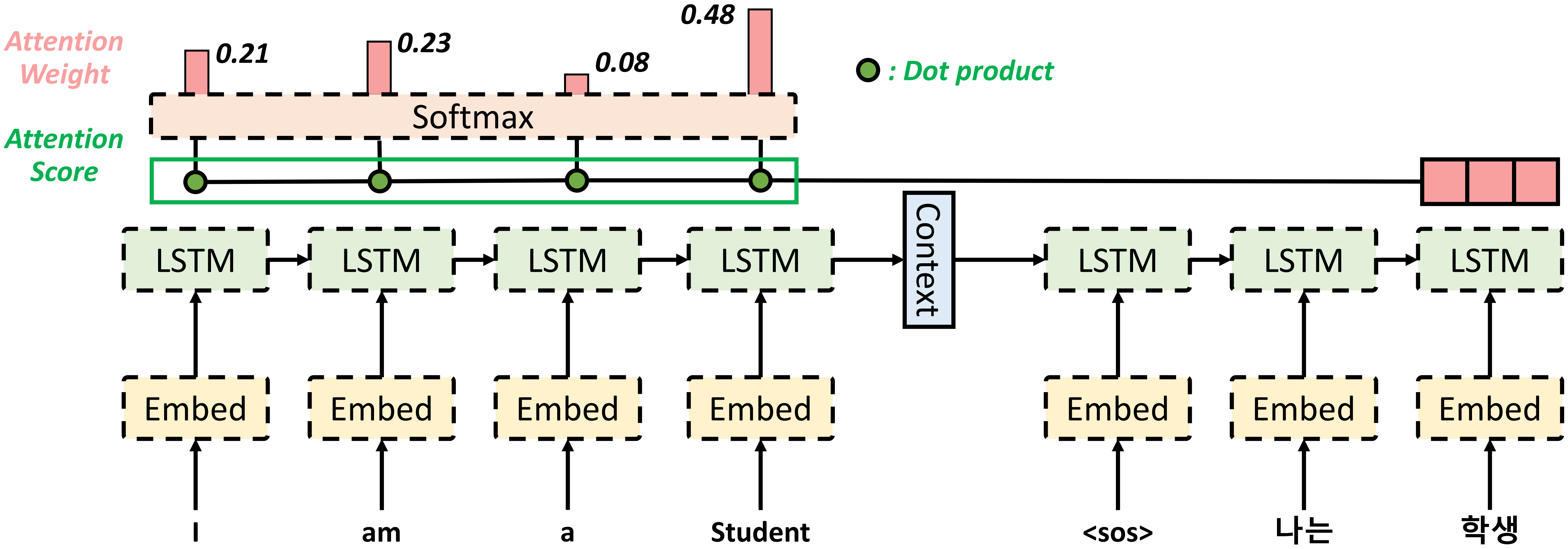
$$score(s_{t-1}, h_i) = s_{t-1}^T h_i$$

$$e^t = [s_{t-1}^T h_0, \dots, s_{t-1}^T h_N]$$

https://wikidocs.net/22893

Calculation for Attention Weight

After calculating attention score, attention weight for ***each hidden state*** of the encoder is calculated by ***softmax function***.

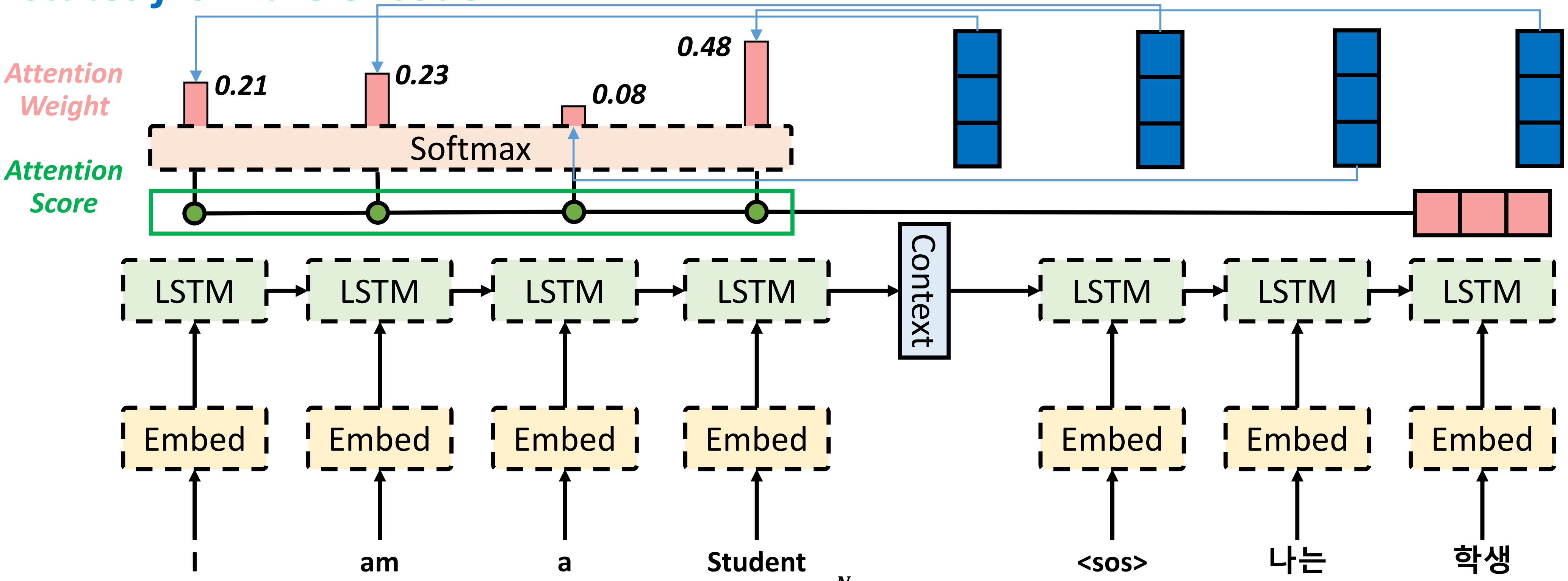


$$\alpha^t = softmax(e^t)$$

https://wikidocs.net/22893

Calculation for Attention Value

Attention value is ***multiplication process*** between ***the attention weight*** and ***hidden states from the encoder***.

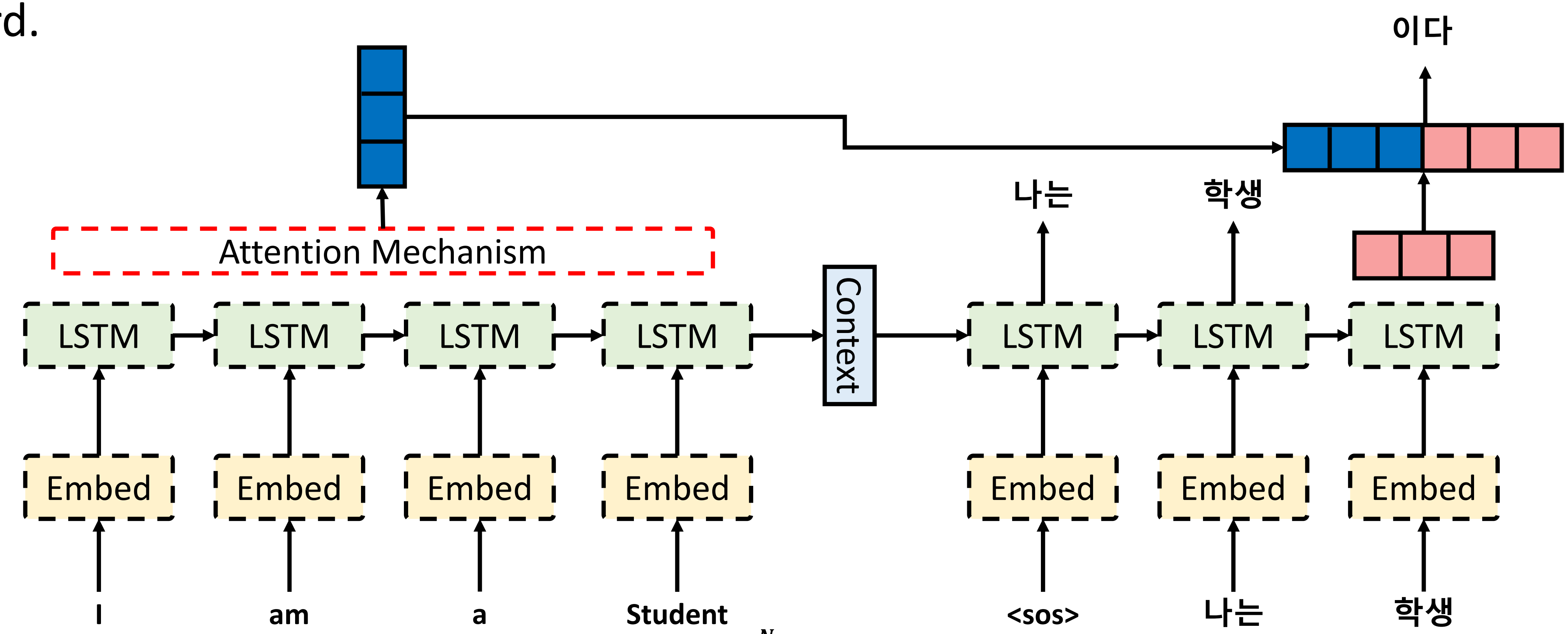


$$\alpha^t = \sum_{i=1}^{N} \alpha_i^t h_i$$

https://wikidocs.net/22893

Concatenation and Prediction

Attention value is ***concatenated with previous hidden state*** in order to predict next word.



$$\boldsymbol{a}^t = \sum_{i=1}^{N} \alpha_i^t \, \boldsymbol{h}_i$$

https://wikidocs.net/22893

Different Type of Attention Mechanism

There are several variant versions of attention mechanism such as using cosine similarity, tanh function and location-based.

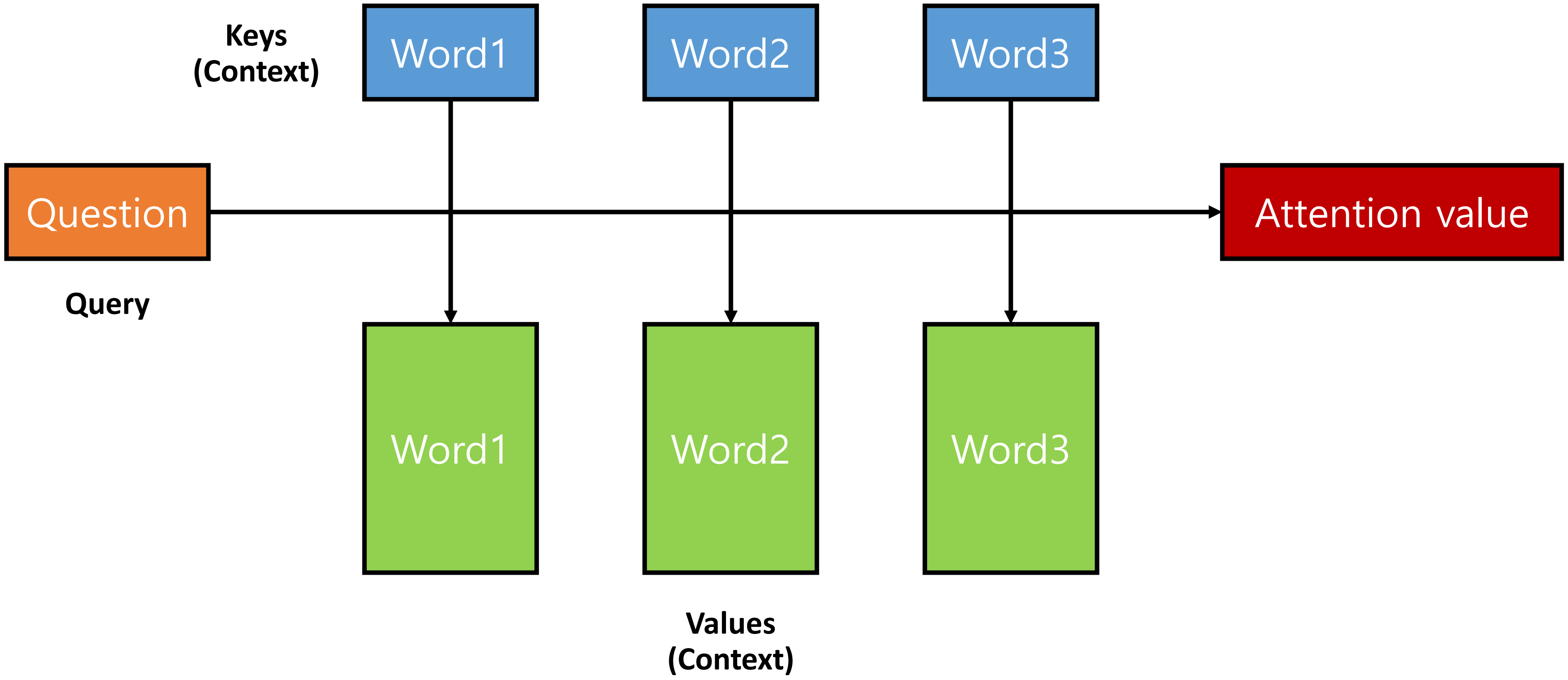
1. Content-base attention: $score(s_i, h_i) = cosine[s_i, h_i]$

2. Additive: $score(s_i, h_i) = v_\alpha^T \tanh(W_\alpha[s_i; h_i]$

3. Location-base: $\alpha_{t,i} = softmax(W_\alpha s_t)$

Attention Mechanism on Question Answering

In question answering task, it is easily applied on QA task since we can ***substitute the inputs into Query, Keys and Values***.



Attention Mechanism on Question Answering

This is an example of (Q, K, V) in visual question answering task.

Image

Keys and Values

Question

What is the ***mustache*** made of?

Query

Attention Mechanism on Question Answering

This is an example of (Q, K, V) in video question answering task.

Sequence of frames $X = [x_1, \dots, x_T]$

Keys and Values

Attention for each keys and values

Query

Question Q

Who enters the coffee shop after Ross shows everyone the paper?

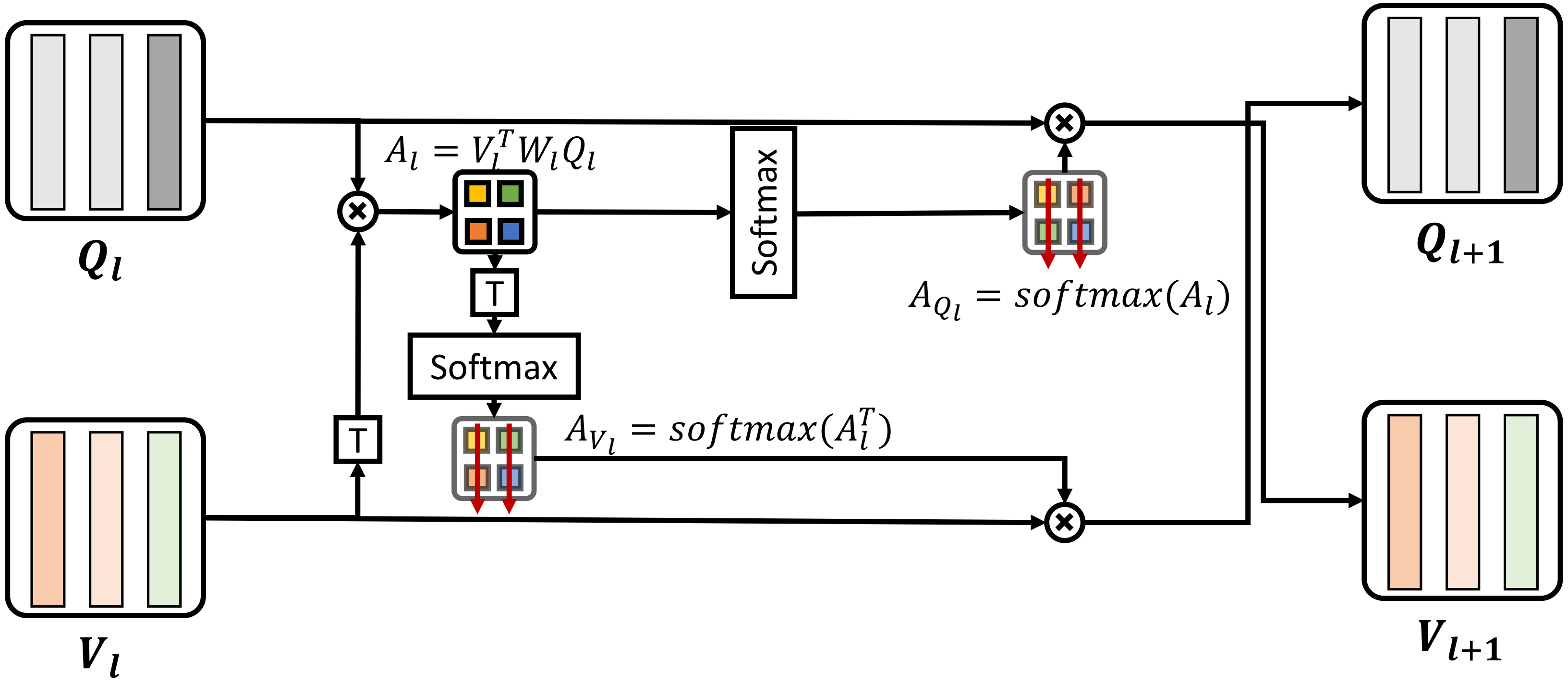
Keys and Values

Answer A

1) Joey 2) Rachel 3) Monica 4) Chandler 5) Phoebe

Bi-directional Attention Mechanism

It applies the attention mechanism ***both directional way for a question and context***.



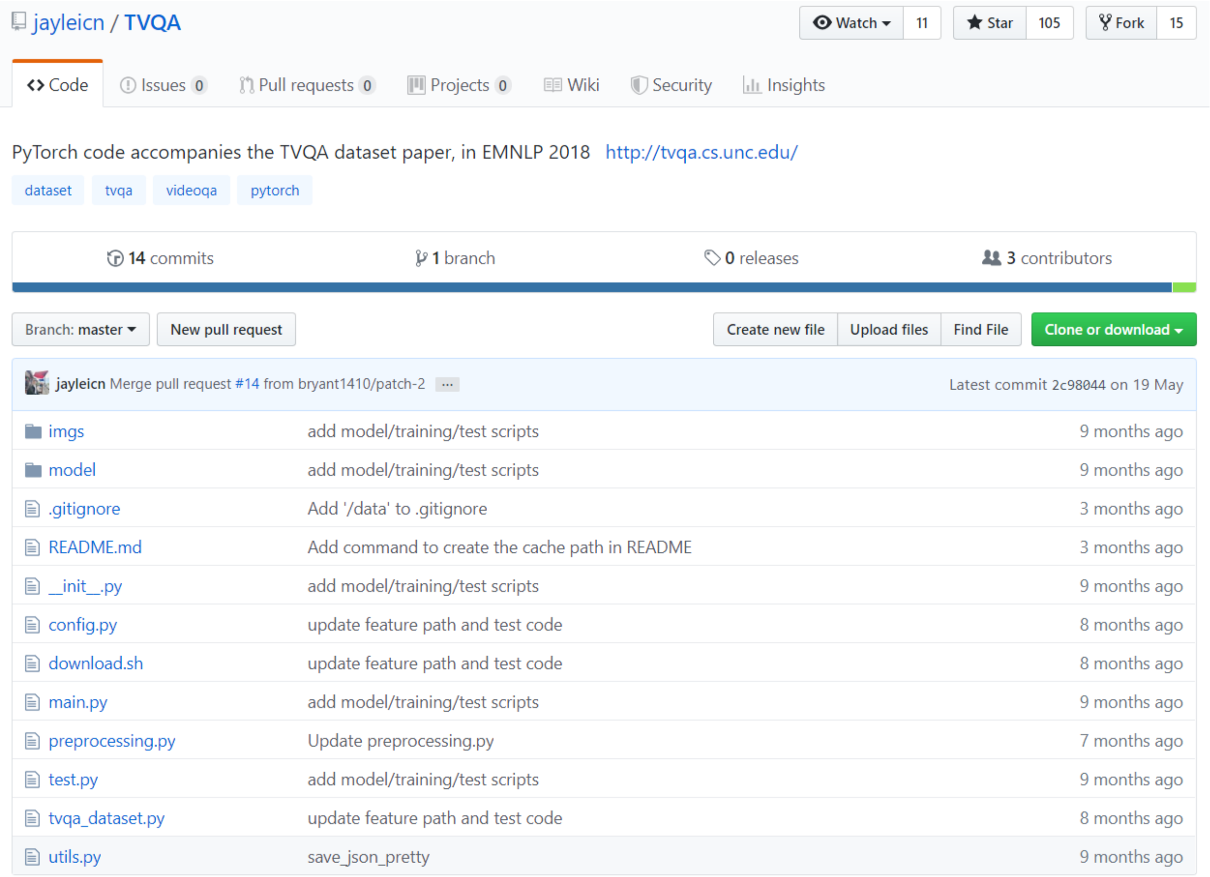
In [Lei18], they used uni-directional attention mechanism due to limitations on memory capacity.

[Nguyen18] D. Nguyen, T. Okatani, Improved Fusion of Visual and Language Representations by Dense Symmetric Co-Attention for Visual Question Answering. CVPR 2018

[Lei18] J. Lei, L. Yu, M. Bansal, T. L. Berg, TVQA: Localized, Compositional Video Question Answering. EMNLP 2018

Code Review

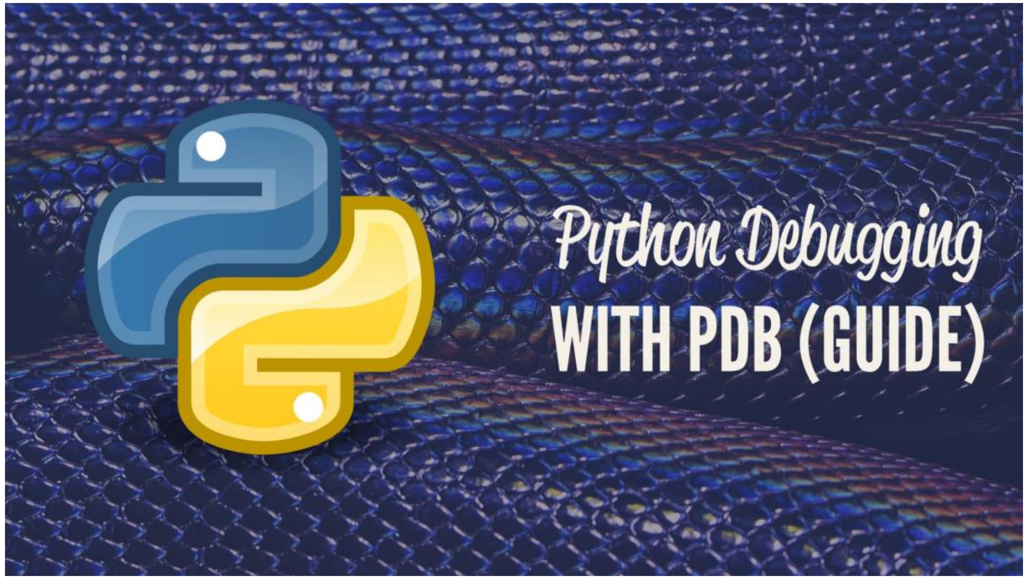
We are going to explore how ***Multi-Modal Video QA*** is written by Python code.

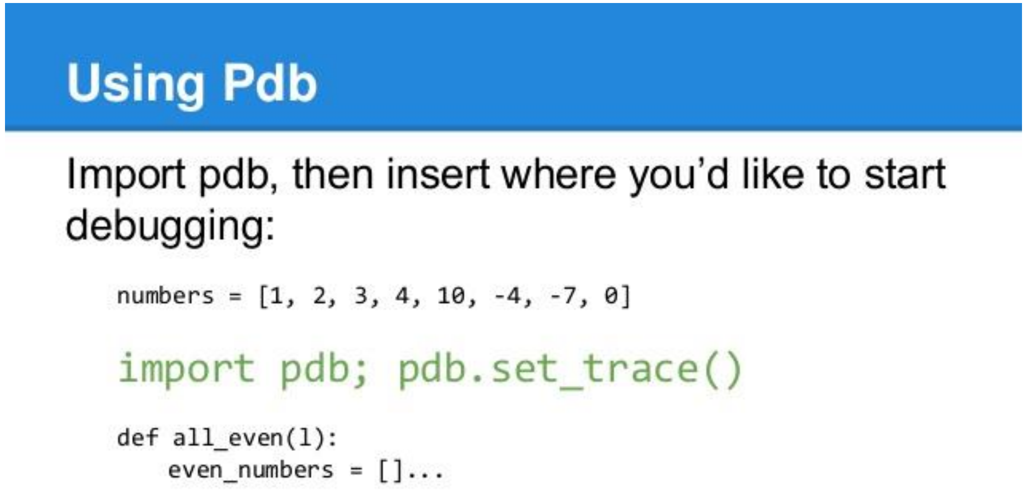


https://github.com/jayleicn/TVQA

Python Debugger (PDB)

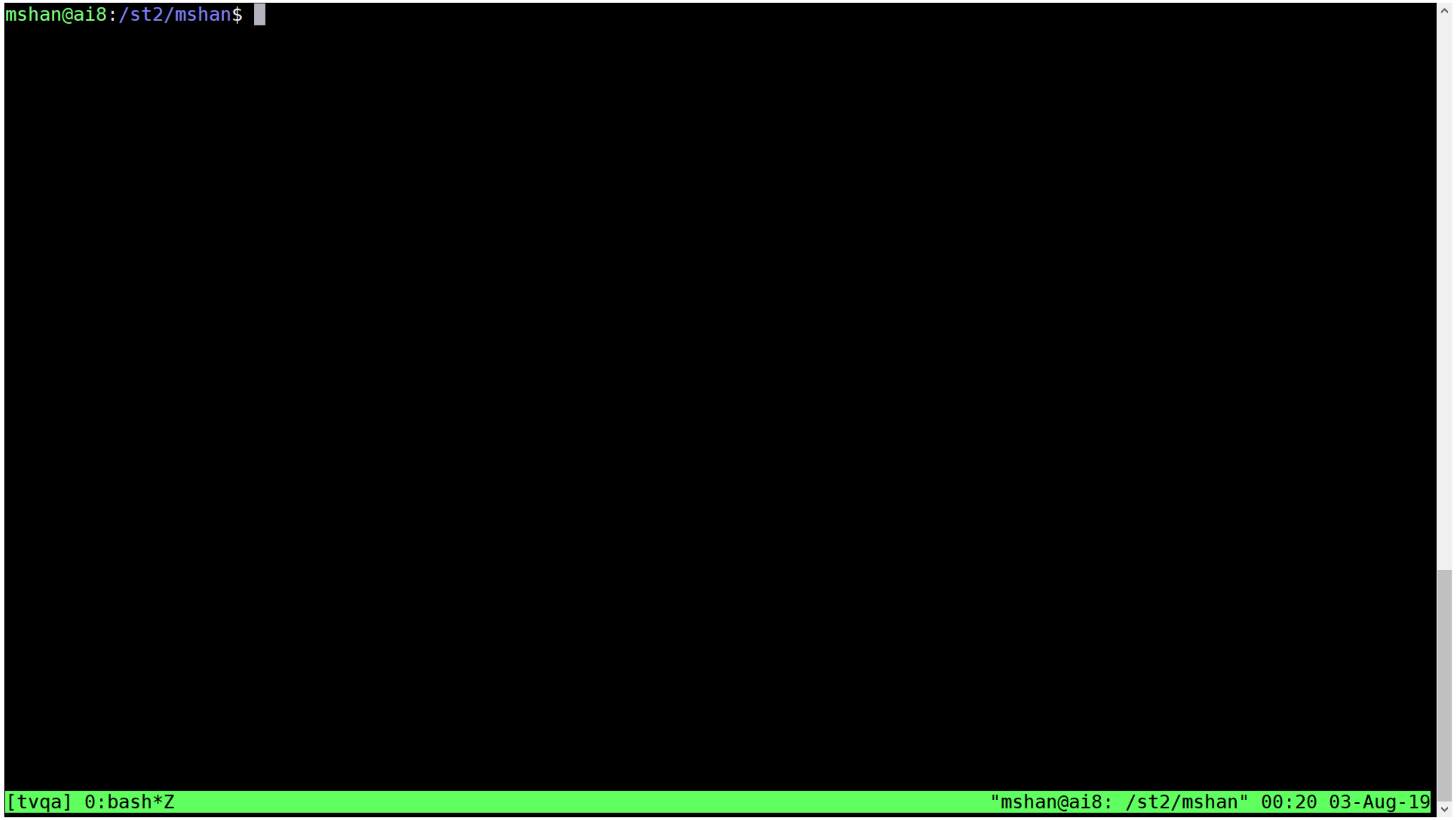
Before exploring, we should know about a useful tool called ‘PDB’ to debug Python code.





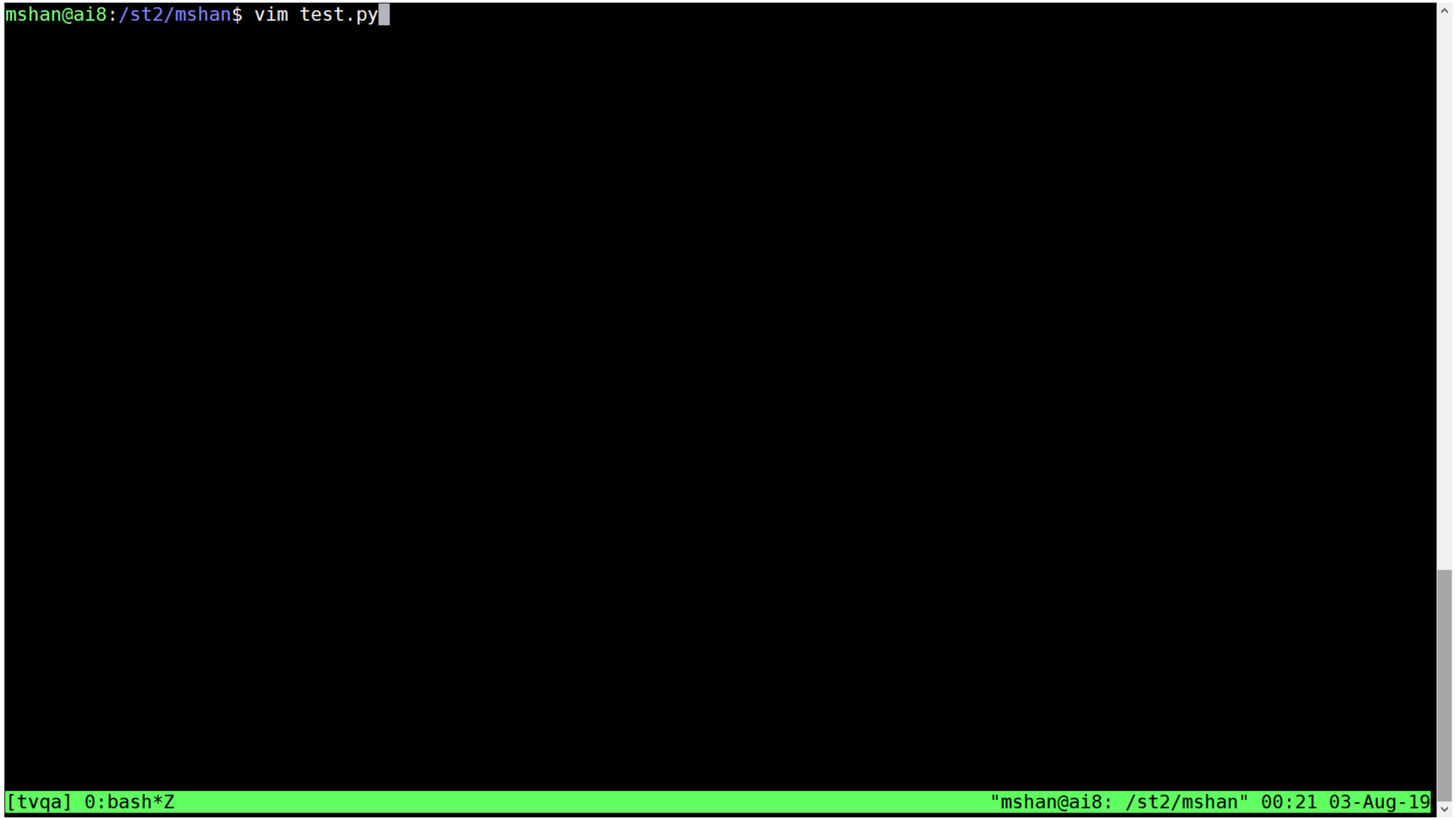
Example of Debugging Process

1. Open a terminal, >> Ctrl + Shift + t



Example of Debugging Process

2. Create a vim file, >> vim test.py



Example of Debugging Process

3. Type below lines on your vim file.

>> i

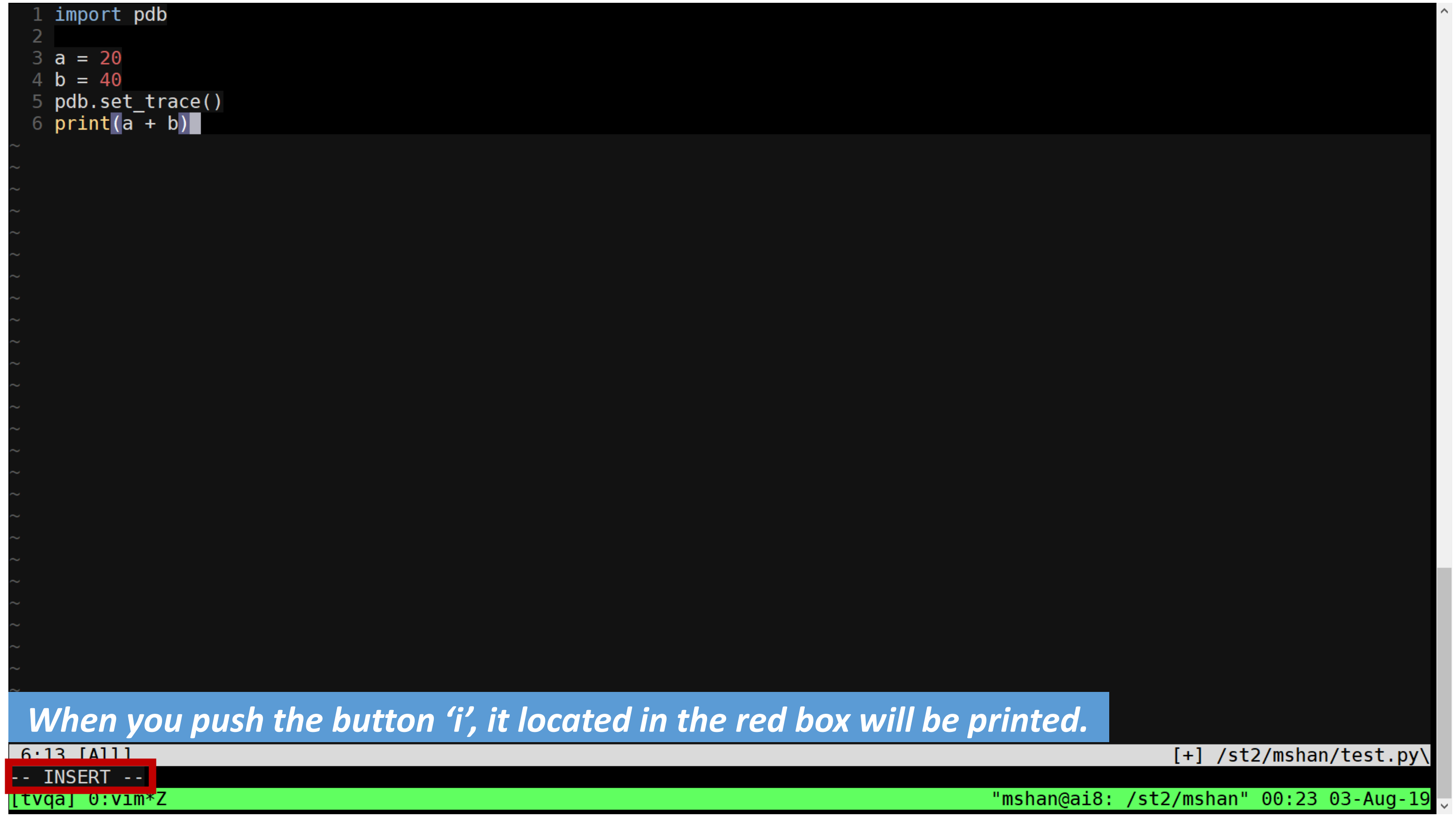
>> import pdb

>> a = 20

>> b = 40

>> pdb.set_trace()

>> print(a + b)



Example of Debugging Process

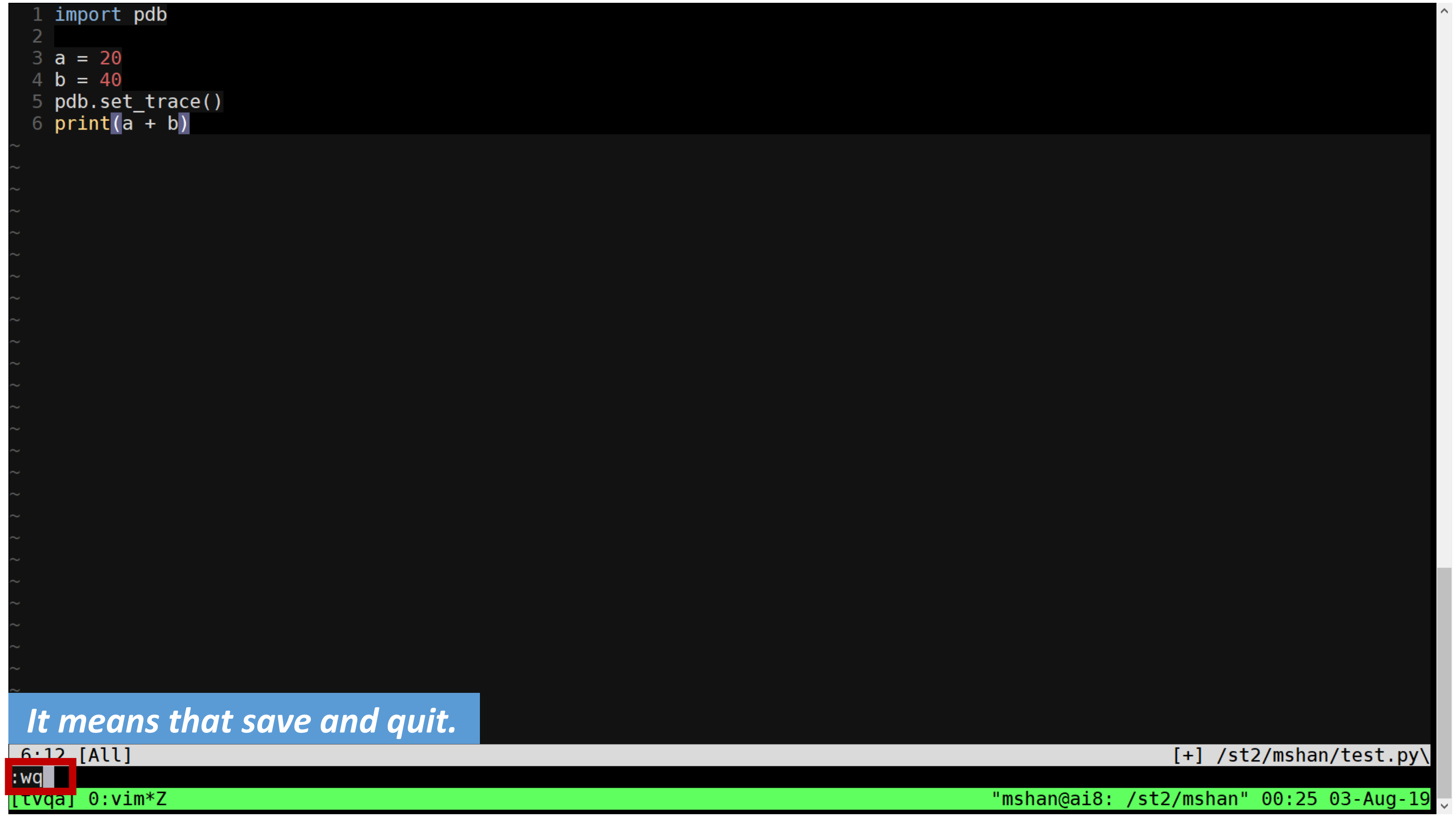
4. Save the vim file and run

>> ESC

>> :

>> wq

>> python3 test.py



Example of Debugging Process

5. The code will be stopped at the code ‘pdb.set_trace()’.

>> n

```
mshan@ai8:/st2/mshan$ python test.py
> /st2/mshan/test.py(6)<module>()
-> print(a + b)
(Pdb) 
```

What result can you see on the screen? It should output 60 by ‘print(a + b)’.

Python Debugger (PDB)

Using this example, we can explore all codes written in Python.

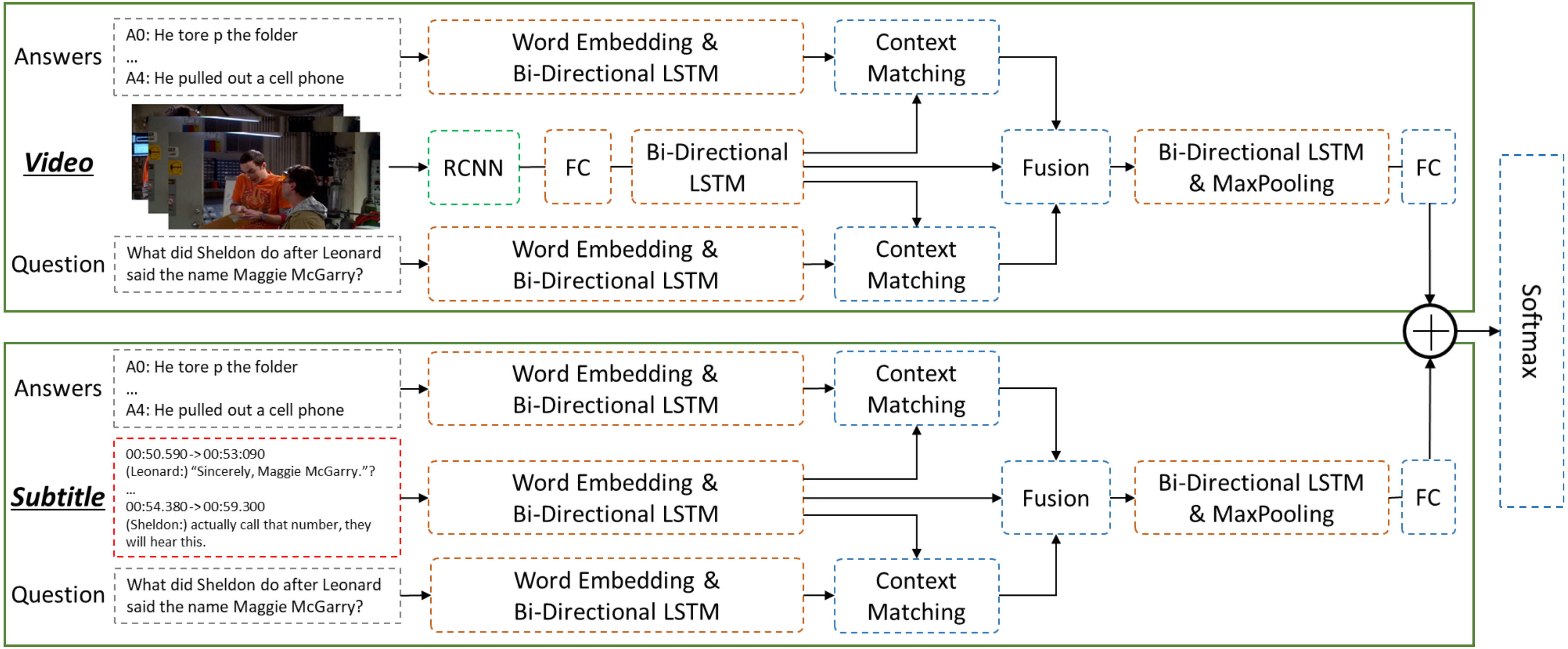
PDB 명령어	실행내용
help	도움말
next	다음 문장으로 이동
print	변수값 화면에 표시
list	소스코드 리스트 출력. 현재 위치 화살표로 표시됨
where	콜스택 출력
continue	계속 실행. 다음 중단점에 멈추거나 중단점 없으면 끝까지 실행
step	Step Into 하여 함수 내부로 들어감
return	현재 함수의 리턴 직전까지 실행
!변수명 = 값	변수에 값 재설정

You can see more information about PDB on https://docs.python.org/3/library/pdb.html

http://pythonstudy.xyz/python/article/505-Python-%EB%94%94%EB%B2%84%EA%B9%85-PDB

Multi-Modal Video QA

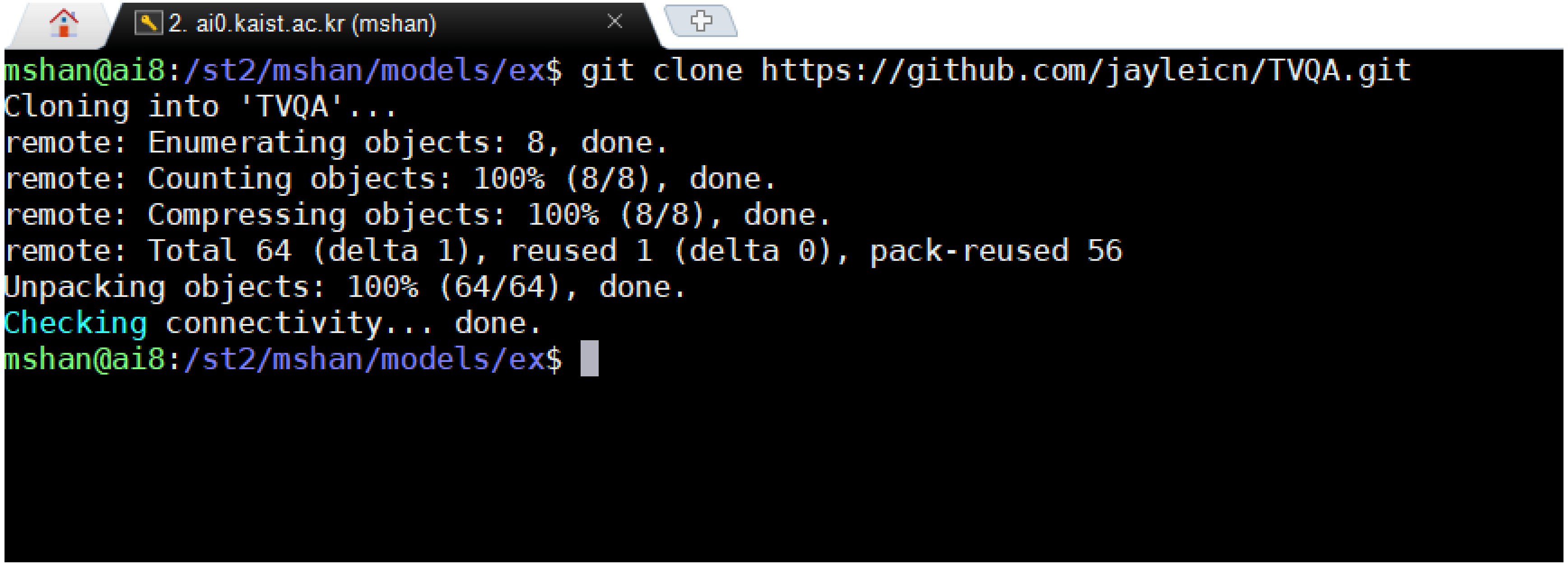
The main components of this model are ***Word Embedding***, ***Bi-directional LSTM***, ***Context Matching*** and ***Fusion layer***.



[Lei18] J. Lei, L. Yu, M. Bansal, T. L. Berg, TVQA: Localized, Compositional Video Question Answering. EMNLP 2018

Download TVQA GitHub

>> git clone https://github.com/jayleicn/TVQA.git



https://github.com/jayleicn/TVQA

Dependency Installation

```
>> pip install torch torchvision
>> pip install h5py
>> pip install tqdm
>> pip install pysrt
>> pip install tensorboardX
>> pip install numpy
```

https://github.com/jayleicn/TVQA

Download TVQA Dataset

>> cd TVQA

>> bash download.sh (or sh download.sh)

```
2. ai0.kaist.ac.kr (mshan)
mshan@ai8:/st2/mshan/models/ex/TVQA$ sh download.sh
[log] Start downloading files
tvqa_qa_release.tar.gz              100%[==========================================================================>]  13.80M  3.59MB/s    in 4.7s
tvqa_subtitles.tar.gz               100%[==========================================================================>]  14.78M  2.62MB/s    in 6.7s
frm_cnt_cache.tar.gz                100%[==========================================================================>]  99.82K   179KB/s    in 0.6s
det_visual_concepts_hq.pickle.tar.gz 100%[=========================================================================>]  96.05M  2.11MB/s    in 48s
md5sum: tvqa_data.md5: No such file or directory
[Log] Found corrupted file, please re-download the files.
mshan@ai8:/st2/mshan/models/ex/TVQA$
```

```
2. ai0.kaist.ac.kr (mshan)
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ ls
det_visual_concepts_hq.pickle.tar.gz  frm_cnt_cache.tar.gz  tvqa_qa_release.tar.gz  tvqa_subtitles.tar.gz
mshan@ai8:/st2/mshan/models/ex/TVQA/data$
```

https://github.com/jayleicn/TVQA

Download GloVe

Move to https://github.com/stanfordnlp/GloVe and download a file in the red box.

Download pre-trained word vectors

The links below contain word vectors obtained from the respective corpora. If you want word vectors trained on massive web datasets, you need only download one of these text files! Pre-trained word vectors are made available under the Public Domain Dedication and License.

- Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
- Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
- Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vectors, 822 MB download): glove.6B.zip
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors, 1.42 GB download): glove.twitter.27B.zip

https://github.com/stanfordnlp/GloVe

Download GloVe

Move ‘glove.6B.zip’ to TVQA/data

2. ai0.kaist.ac.kr (mshan)

```
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ ls
det_visual_concepts_hq.pickle.tar.gz  frm_cnt_cache.tar.gz  glove.6B.zip  tvqa_qa_release.tar.gz  tvqa_subtitles.tar.gz
mshan@ai8:/st2/mshan/models/ex/TVQA/data$
```

https://github.com/stanfordnlp/GloVe

Decompression for GloVe

>> cd TVQA/data

>> unzip glove.6B.zip

```
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ tar -zxvf det_visual_concepts_hq.pickle.tar.gz
det_visual_concepts_hq.pickle
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ unzip glove.6B.zip
Archive:  glove.6B.zip
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
  inflating: glove.6B.50d.txt
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ ls
det_visual_concepts_hq.pickle         frm_cnt_cache.tar.gz  glove.6B.200d.txt  glove.6B.50d.txt  tvqa_qa_release.tar.gz
det_visual_concepts_hq.pickle.tar.gz  glove.6B.100d.txt     glove.6B.300d.txt  glove.6B.zip      tvqa_subtitles.tar.gz
mshan@ai8:/st2/mshan/models/ex/TVQA/data$
```

https://github.com/stanfordnlp/GloVe

Decompression for TVQA Dataset

>> cd TVQA/data

>> tar -zxvf tvqa_qa_release.tar.gz

>> tar -zxvf tvqa_subtitles.tar.gz

>> tar -zxvf frm_cnt_cache.tar.gz

```
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ tar -zxvf tvqa_qa_release.tar.gz
tvqa_qa_release/
tvqa_qa_release/tvqa_train.jsonl
tvqa_qa_release/tvqa_val.jsonl
tvqa_qa_release/tvqa_test_public.jsonl
```

```
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ tar -zxvf tvqa_subtitles.tar.gz
```

```
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ tar -zxvf frm_cnt_cache.tar.gz
frm_cnt_cache.json
mshan@ai8:/st2/mshan/models/ex/TVQA/data$
```

```
mshan@ai8:/st2/mshan/models/ex/TVQA/data$ ls
det_visual_concepts_hq.pickle         frm_cnt_cache.tar.gz   glove.6B.300d.txt   srt_data_cache.json      tvqa_subtitles
det_visual_concepts_hq.pickle.tar.gz  glove.6B.100d.txt      glove.6B.50d.txt    tvqa_qa_release          tvqa_subtitles.tar.gz
frm_cnt_cache.json                    glove.6B.200d.txt      glove.6B.zip        tvqa_qa_release.tar.gz
mshan@ai8:/st2/mshan/models/ex/TVQA/data$
```

https://github.com/jayleicn/TVQA

Pre-processing for TVQA Dataset

>> preprocessing.py

```
mshan@ai8:/st2/mshan/models/ex/TVQA$ python preprocessing.py
Loading srt files from ./data/tvqa_subtitles ...
100%|██████████| 21793/21793 [01:10<00:00, 309.26it/s]
Tokenize subtitle ...
100%|██████████| 21793/21793 [00:24<00:00, 889.08it/s]
------------------------------------------------------------
Processing ./data/tvqa_qa_release/tvqa_val.jsonl
Tokenize QA ...
100%|██████████| 15253/15253 [00:04<00:00, 3756.89it/s]
Adding subtitle ...
100%|██████████| 15253/15253 [00:00<00:00, 99078.25it/s]
Found frame cnt cache, loading ...
100%|██████████| 15253/15253 [00:00<00:00, 20818.64it/s]
There are 6 NaN values in ts, which are replaced by [10, 30], will be fixed later
------------------------------------------------------------
Processing ./data/tvqa_qa_release/tvqa_test_public.jsonl
Tokenize QA ...
100%|██████████| 7623/7623 [00:02<00:00, 3671.02it/s]
Adding subtitle ...
100%|██████████| 7623/7623 [00:00<00:00, 65175.57it/s]
Found frame cnt cache, loading ...
100%|██████████| 7623/7623 [00:00<00:00, 19019.46it/s]
There are 3 NaN values in ts, which are replaced by [10, 30], will be fixed later
------------------------------------------------------------
Processing ./data/tvqa_qa_release/tvqa_train.jsonl
Tokenize QA ...
100%|██████████| 122039/122039 [00:33<00:00, 3684.69it/s]
Adding subtitle ...
100%|██████████| 122039/122039 [00:01<00:00, 106054.20it/s]
Found frame cnt cache, loading ...
100%|██████████| 122039/122039 [00:07<00:00, 15454.28it/s]
There are 36 NaN values in ts, which are replaced by [10, 30], will be fixed later
mshan@ai8:/st2/mshan/models/ex/TVQA$
```

! Notice

Original code provided by TVQA is based on Python2.

Therefore, you must convert Python2 into Python3.

→ See the code installed on your desktop, which is already converted.

https://github.com/jayleicn/TVQA

Pre-processing for TVQA Dataset

>> cd TVQA

>> mkdir cache

>> python tvqa_dataset.py

```
mshan@ai8:/st2/mshan/models/ex/TVQA$ mkdir cache
mshan@ai8:/st2/mshan/models/ex/TVQA$ python tvqa_dataset.py
------------ Options -------------
bsz: 32
debug: False
device: 0
embedding_size: 300
glove_path: ./data/glove.6B.300d.txt
hsz1: 150
hsz2: 300
idx2word_path: ./cache/idx2word.pickle
input_streams: ['sub']
log_freq: 400
lr: 0.0003
max_es_cnt: 3
max_sub_l: 300
max_vcpt_l: 300
max_vid_l: 480
```

! Notice

Original code provided by TVQA is based on Python2.

Therefore, you must convert Python2 into Python3.

→ See the code installed on your desktop, which is already converted.

https://github.com/jayleicn/TVQA

Run

>> cd TVQA

>> python main.py --input_streams sub vcpt

```
mshan@ai8:/st2/mshan/models/ex/TVQA$ python main.py --input_stream sub vcpt
------------ Options -------------
bsz: 32
debug: False
device: 0
embedding_size: 300
glove_path: ./data/glove.6B.300d.txt
hsz1: 150
hsz2: 300
idx2word_path: ./cache/idx2word.pickle
input_streams: ['sub', 'vcpt']
log_freq: 400
lr: 0.0003
max_es_cnt: 3
max_sub_l: 300
max_vcpt_l: 300
max_vid_l: 480
n_epoch: 100
n_layers_cls: 1
no_core_driver: False
no_glove: False
no_normalize_v: False
no_ts: False
results_dir_base: results/results
test_bsz: 100
test_path: ./data/tvqa_test_public_processed.json
train_path: ./data/tvqa_train_processed.json
valid_path: ./data/tvqa_val_processed.json
vcpt_path: ./data/det_visual_concepts_hq.pickle
vid_feat_path: ./data/tvqa_imagenet_pool5.h5
vid_feat_size: 2048
vocab_embedding_path: ./cache/vocab_embedding.pickle
vocab_size: 0
wd: 1e-05
word2idx_path: ./cache/word2idx.pickle
word_count_threshold: 2
-------------- End ----------------
```

```
word_count_threshold: 2
-------------- End ----------------

Loading cache ...
activate sub stream
activate vcpt stream
/home/mshan/.local/lib/python2.7/site-packages/torch/nn/_reduction.py:46: UserWarning: size_average and r
' instead.
  warnings.warn(warning.format(ret))
0it [00:00, ?it/s] Train Epoch 0 loss 1.6104 acc 0.0938 Val loss 1.6093 acc 0.2092
52it [03:11,  1.10it/s]
```

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```python
if __name__ == "__main__":
    torch.manual_seed(2018)
    opt = BaseOptions().parse()
    writer = SummaryWriter(opt.results_dir)
    opt.writer = writer

    dset = TVQADataset(opt)
    opt.vocab_size = len(dset.word2idx)
    model = ABC(opt)
    if not opt.no_glove:
        model.load_embedding(dset.vocab_embedding)

    model.to(opt.device)
    cudnn.benchmark = True
    criterion = nn.CrossEntropyLoss(size_average=False).to(opt.device)
    optimizer = torch.optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),
                                 lr=opt.lr, weight_decay=opt.wd)

    best_acc = 0.
    early_stopping_cnt = 0
    early_stopping_flag = False
    for epoch in range(opt.n_epoch):
        if not early_stopping_flag:
            # train for one epoch, valid per n batches, save the log and the best model
            cur_acc = train(opt, dset, model, criterion, optimizer, epoch, best_acc)

            # remember best acc
            is_best = cur_acc > best_acc
            best_acc = max(cur_acc, best_acc)
            if not is_best:
                early_stopping_cnt += 1
                if early_stopping_cnt >= opt.max_es_cnt:
                    early_stopping_flag = True
        else:
            print("early stop with valid acc %.4f" % best_acc)
            opt.writer.export_scalars_to_json(os.path.join(opt.results_dir, "all_scalars.json"))
            opt.writer.close()
            break  # early stop break
```

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```
if __name__ == "__main__":
    torch.manual_seed(2018)
    opt = BaseOptions().parse()
    writer = SummaryWriter(opt.results_dir)
    opt.writer = writer

    dset = TVQADataset(opt)
    opt.vocab_size = len(dset.word2idx)
    model = ABC(opt)
    if not opt.no_glove:
        model.load_embedding(dset.vocab_embedding)
```

Initialize seed

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```
if __name__ == "__main__":
    torch.manual_seed(2018)
    opt = BaseOptions().parse()
    writer = SummaryWriter(opt.results_dir)
    opt.writer = writer

    dset = TVQADataset(opt)
    opt.vocab_size = len(dset.word2idx)
    model = ABC(opt)
    if not opt.no_glove:
        model.load_embedding(dset.vocab_embedding)
```

Load arguments

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```
if __name__ == "__main__":
    torch.manual_seed(2018)
    opt = BaseOptions().parse()
    writer = SummaryWriter(opt.results_dir)
    opt.writer = writer

    dset = TVQADataset(opt)
    opt.vocab_size = len(dset.word2idx)
    model = ABC(opt)
    if not opt.no_glove:
        model.load_embedding(dset.vocab_embedding)
```

Initialize Tensorboard

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```
if __name__ == "__main__":
    torch.manual_seed(2018)
    opt = BaseOptions().parse()
    writer = SummaryWriter(opt.results_dir)
    opt.writer = writer

    dset = TVQADataset(opt)
    opt.vocab_size = len(dset.word2idx)
    model = ABC(opt)
    if not opt.no_glove:
        model.load_embedding(dset.vocab_embedding)
```

Load pre-processed dataset

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```
if __name__ == "__main__":
    torch.manual_seed(2018)
    opt = BaseOptions().parse()
    writer = SummaryWriter(opt.results_dir)
    opt.writer = writer

    dset = TVQADataset(opt)
    opt.vocab_size = len(dset.word2idx)
    model = ABC(opt)
    if not opt.no_glove:
        model.load_embedding(dset.vocab_embedding)
```

Define the size of vocabulary

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```
if __name__ == "__main__":
    torch.manual_seed(2018)
    opt = BaseOptions().parse()
    writer = SummaryWriter(opt.results_dir)
    opt.writer = writer

    dset = TVQADataset(opt)
    opt.vocab_size = len(dset.word2idx)
    model = ABC(opt)
    if not opt.no_glove:
        model.load_embedding(dset.vocab_embedding)
```

Load model

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```python
if __name__ == "__main__":
    torch.manual_seed(2018)
    opt = BaseOptions().parse()
    writer = SummaryWriter(opt.results_dir)
    opt.writer = writer

    dset = TVQADataset(opt)
    opt.vocab_size = len(dset.word2idx)
    model = ABC(opt)
    if not opt.no_glove:
        model.load_embedding(dset.vocab_embedding)
```

GloVe initialization

https://github.com/jayleicn/TVQA

Main

>> vim main.py

```
model.to(opt.device)
cudnn.benchmark = True
criterion = nn.CrossEntropyLoss(size_average=False).to(opt.device)
optimizer = torch.optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),
                             lr=opt.lr, weight_decay=opt.wd)
```

Define loss function and optimizer

https://github.com/jayleicn/TVQA

Main

>> vim main.py

Training process

```
best_acc = 0.
early_stopping_cnt = 0
early_stopping_flag = False
for epoch in range(opt.n_epoch):
    if not early_stopping_flag:
        # train for one epoch, valid per n batches, save the log and the best model
        cur_acc = train(opt, dset, model, criterion, optimizer, epoch, best_acc)

        # remember best acc
        is_best = cur_acc > best_acc
        best_acc = max(cur_acc, best_acc)
        if not is_best:
            early_stopping_cnt += 1
            if early_stopping_cnt >= opt.max_es_cnt:
                early_stopping_flag = True
    else:
        print("early stop with valid acc %.4f" % best_acc)
        opt.writer.export_scalars_to_json(os.path.join(opt.results_dir, "all_scalars.json"))
        opt.writer.close()
        break  # early stop break
```

https://github.com/jayleicn/TVQA

Train

>> vim main.py

```
def train(opt, dset, model, criterion, optimizer, epoch, previous_best_acc):
    dset.set_mode("train")
    model.train()
    train_loader = DataLoader(dset, batch_size=opt.bsz, shuffle=True, collate_fn=pad_collate)

    train_loss = []
    valid_acc_log = ["batch_idx\tacc"]
    train_corrects = []
    torch.set_grad_enabled(True)
    for batch_idx, batch in tqdm(enumerate(train_loader)):
        model_inputs, targets, _ = preprocess_inputs(batch, opt.max_sub_l, opt.max_vcpt_l, opt.max_vid_l,
                                                     device=opt.device)
        outputs = model(*model_inputs)
        loss = criterion(outputs, targets)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # measure accuracy and record loss
        train_loss.append(loss.item())
        pred_ids = outputs.data.max(1)[1]
        train_corrects += pred_ids.eq(targets.data).cpu().numpy().tolist()
        if batch_idx % opt.log_freq == 0:
            niter = epoch * len(train_loader) + batch_idx

            train_acc = sum(train_corrects) / float(len(train_corrects))
            train_loss = sum(train_loss) / float(len(train_corrects))
            opt.writer.add_scalar("Train/Acc", train_acc, niter)
            opt.writer.add_scalar("Train/Loss", train_loss, niter)

            # Test
            valid_acc, valid_loss = validate(opt, dset, model, mode="valid")
            opt.writer.add_scalar("Valid/Loss", valid_loss, niter)

            valid_log_str = "%02d\t%.4f" % (batch_idx, valid_acc)
            valid_acc_log.append(valid_log_str)
            if valid_acc > previous_best_acc:
                previous_best_acc = valid_acc
                torch.save(model.state_dict(), os.path.join(opt.results_dir, "best_valid.pth"))
            print(" Train Epoch %d loss %.4f acc %.4f Val loss %.4f acc %.4f"
```

https://github.com/jayleicn/TVQA

Train

>> vim main.py

```
def train(opt, dset, model, criterion, optimizer, epoch, previous_best_acc):
    dset.set_mode("train")
    model.train()
    train_loader = DataLoader(dset, batch_size=opt.bsz, shuffle=True, collate_fn=pad_collate)

    train_loss = []
    valid_acc_log = ["batch_idx\tacc"]
    train_corrects = []
    torch.set_grad_enabled(True)
    for batch_idx, batch in tqdm(enumerate(train_loader)):
        model_inputs, targets, _ = preprocess_inputs(batch, opt.max_sub_l, opt.max_vcpt_l, opt.max_vid_l,
                                                     device=opt.device)
        outputs = model(*model_inputs)
        loss = criterion(outputs, targets)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Forward and Backward

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
self.parser.add_argument("--debug", action="store_true", help="debug mode, break all loops")
self.parser.add_argument("--results_dir_base", type=str, default="results/results")
self.parser.add_argument("--log_freq", type=int, default=400, help="print, save training info")
self.parser.add_argument("--lr", type=float, default=3e-4, help="learning rate")
self.parser.add_argument("--wd", type=float, default=1e-5, help="weight decay")
self.parser.add_argument("--n_epoch", type=int, default=100, help="number of epochs to run")
self.parser.add_argument("--max_es_cnt", type=int, default=3, help="number of epochs to early stop")
self.parser.add_argument("--bsz", type=int, default=32, help="mini-batch size")
self.parser.add_argument("--test_bsz", type=int, default=100, help="mini-batch size for testing")
self.parser.add_argument("--device", type=int, default=0, help="gpu ordinal, -1 indicates cpu")
self.parser.add_argument("--no_core_driver", action="store_true",
                         help="hdf5 driver, default use `core` (load into RAM), if specified, use `None`")
self.parser.add_argument("--word_count_threshold", type=int, default=2, help="word vocabulary threshold")
```

Learning rate

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
self.parser.add_argument("--debug", action="store_true", help="debug mode, break all loops")
self.parser.add_argument("--results_dir_base", type=str, default="results/results")
self.parser.add_argument("--log_freq", type=int, default=400, help="print, save training info")
self.parser.add_argument("--lr", type=float, default=3e-4, help="learning rate")
self.parser.add_argument("--wd", type=float, default=1e-5, help="weight decay")
self.parser.add_argument("--n_epoch", type=int, default=100, help="number of epochs to run")
self.parser.add_argument("--max_es_cnt", type=int, default=3, help="number of epochs to early stop")
self.parser.add_argument("--bsz", type=int, default=32, help="mini-batch size")
self.parser.add_argument("--test_bsz", type=int, default=100, help="mini-batch size for testing")
self.parser.add_argument("--device", type=int, default=0, help="gpu ordinal, -1 indicates cpu")
self.parser.add_argument("--no_core_driver", action="store_true",
                         help="hdf5 driver, default use `core` (load into RAM), if specified, use `None`")
self.parser.add_argument("--word_count_threshold", type=int, default=2, help="word vocabulary threshold")
```

Weight decay on Adam optimizer

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
self.parser.add_argument("--debug", action="store_true", help="debug mode, break all loops")
self.parser.add_argument("--results_dir_base", type=str, default="results/results")
self.parser.add_argument("--log_freq", type=int, default=400, help="print, save training info")
self.parser.add_argument("--lr", type=float, default=3e-4, help="learning rate")
self.parser.add_argument("--wd", type=float, default=1e-5, help="weight decay")
self.parser.add_argument("--n_epoch", type=int, default=100, help="number of epochs to run")
self.parser.add_argument("--max_es_cnt", type=int, default=3, help="number of epochs to early
self.parser.add_argument("--bsz", type=int, default=32, help="mini-batch size")
self.parser.add_argument("--test_bsz", type=int, default=100, help="mini-batch size for testing")
self.parser.add_argument("--device", type=int, default=0, help="gpu ordinal, -1 indicates cpu")
self.parser.add_argument("--no_core_driver", action="store_true",
                         help="hdf5 driver, default use `core` (load into RAM), if specified, use `None`")
self.parser.add_argument("--word_count_threshold", type=int, default=2, help="word vocabulary threshold")
```

The number of epoch

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
self.parser.add_argument("--debug", action="store_true", help="debug mode, break all loops")
self.parser.add_argument("--results_dir_base", type=str, default="results/results")
self.parser.add_argument("--log_freq", type=int, default=400, help="print, save training info")
self.parser.add_argument("--lr", type=float, default=3e-4, help="learning rate")
self.parser.add_argument("--wd", type=float, default=1e-5, help="weight decay")
self.parser.add_argument("--n_epoch", type=int, default=100, help="number of epochs to run")
self.parser.add_argument("--max_es_cnt", type=int, default=3, help="number of epochs to early
self.parser.add_argument("--bsz", type=int, default=32, help="mini-batch size")
self.parser.add_argument("--test_bsz", type=int, default=100, help="mini-batch size for testi
self.parser.add_argument("--device", type=int, default=0, help="gpu ordinal, -1 indicates cpu")
self.parser.add_argument("--no_core_driver", action="store_true",
                         help="hdf5 driver, default use `core` (load into RAM), if specified, use `None`")
self.parser.add_argument("--word_count_threshold", type=int, default=2, help="word vocabulary threshold")
```

Batch size for each iteration

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
self.parser.add_argument("--debug", action="store_true", help="debug mode, break all loops")
self.parser.add_argument("--results_dir_base", type=str, default="results/results")
self.parser.add_argument("--log_freq", type=int, default=400, help="print, save training info")
self.parser.add_argument("--lr", type=float, default=3e-4, help="learning rate")
self.parser.add_argument("--wd", type=float, default=1e-5, help="weight decay")
self.parser.add_argument("--n_epoch", type=int, default=100, help="number of epochs to run")
self.parser.add_argument("--max_es_cnt", type=int, default=3, help="number of epochs to early stop")
self.parser.add_argument("--bsz", type=int, default=32, help="mini-batch size")
self.parser.add_argument("--test_bsz", type=int, default=100, help="mini-batch size for testing")
self.parser.add_argument("--device", type=int, default=0, help="g
self.parser.add_argument("--no_core_driver", action="store_true",
                         help="hdf5 driver, default use `core` (load into RAM), if specified, use None")
self.parser.add_argument("--word_count_threshold", type=int, default=2, help="word vocabulary threshold")
```

It will be <UNK> token when the number of count of a word is lower than this parameter

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# model config
self.parser.add_argument("--no_glove", action="store_true", help="not use glove vectors")
self.parser.add_argument("--no_ts", action="store_true", help="no
self.parser.add_argument("--input_streams", type=str, nargs="+", choices=["vcpt", "sub", "imagenet"],
                         help="input streams for the model, will use both `vcpt` and `sub` streams")
self.parser.add_argument("--n_layers_cls", type=int, default=1, help="number of layers in classifier")
self.parser.add_argument("--hsz1", type=int, default=150, help="hidden size for the first lstm")
self.parser.add_argument("--hsz2", type=int, default=300, help="hidden size for the second lstm")
self.parser.add_argument("--embedding_size", type=int, default=300, help="word embedding dim")
self.parser.add_argument("--max_sub_l", type=int, default=300, help="max length for subtitle")
self.parser.add_argument("--max_vcpt_l", type=int, default=300, help="max length for visual concepts")
self.parser.add_argument("--max_vid_l", type=int, default=480, help="max length for video feature")
self.parser.add_argument("--vocab_size", type=int, default=0, help="vocabulary size")
self.parser.add_argument("--no_normalize_v", action="store_true", help="do not normalize video featrue")
```

Not used GloVe initialization for embedding layer

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# model config
self.parser.add_argument("--no_glove", action="store_true", help="not use glove vectors")
self.parser.add_argument("--no_ts", action="store_true", help="no timestep annotation, use full length feature")
self.parser.add_argument("--input_streams", type=str, na
                         help="input streams for the mod
self.parser.add_argument("--n_layers_cls", type=int, def
self.parser.add_argument("--hsz1", type=int, default=150, help="hidden size for the first lstm")
self.parser.add_argument("--hsz2", type=int, default=300, help="hidden size for the second lstm")
self.parser.add_argument("--embedding_size", type=int, default=300, help="word embedding dim")
self.parser.add_argument("--max_sub_l", type=int, default=300, help="max length for subtitle")
self.parser.add_argument("--max_vcpt_l", type=int, default=300, help="max length for visual concepts")
self.parser.add_argument("--max_vid_l", type=int, default=480, help="max length for video feature")
self.parser.add_argument("--vocab_size", type=int, default=0, help="vocabulary size")
self.parser.add_argument("--no_normalize_v", action="store_true", help="do not normalize video featrue")
```

All frames in a clip will be used when no_ts is True,
Otherwise used specific frames provided by TVQA dataset

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# model config
self.parser.add_argument("--no_glove", action="store_true", help="not use glove vectors")
self.parser.add_argument("--no_ts", action="store_true", help="no timestep annotation, use full length feature")
self.parser.add_argument("--input_streams", type=str, nargs="+", choices=["vcpt", "sub", "imagenet"],
                         help="input streams for the model, will use both `vcpt` and `sub` streams")
self.parser.add_argument("--n_layers_cls", type=int, default=1,
self.parser.add_argument("--hsz1", type=int, default=150, help=
self.parser.add_argument("--hsz2", type=int, default=300, help=
self.parser.add_argument("--embedding_size", type=int, default=300, help="word embedding dim")
self.parser.add_argument("--max_sub_l", type=int, default=300, help="max length for subtitle")
self.parser.add_argument("--max_vcpt_l", type=int, default=300, help="max length for visual concepts")
self.parser.add_argument("--max_vid_l", type=int, default=480, help="max length for video feature")
self.parser.add_argument("--vocab_size", type=int, default=0, help="vocabulary size")
self.parser.add_argument("--no_normalize_v", action="store_true", help="do not normalize video featrue")
```

Parameters for using the subtitle, visual concepts features, ImageNet features

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# model config
self.parser.add_argument("--no_glove", action="store_true", help="not use glove vectors")
self.parser.add_argument("--no_ts", action="store_true", help="no timestep annotation, use full length feature")
self.parser.add_argument("--input_streams", type=str, nargs="+", choices=["vcpt", "sub", "imagenet"],
                         help="input streams for the model, will use both `vcpt` and `sub` streams")
self.parser.add_argument("--n_layers_cls", type=int, default=1, help="number of layers in classifier")
self.parser.add_argument("--hsz1", type=int, default=150, help="hidden size for the first lstm")
self.parser.add_argument("--hsz2", type=int, default=300, help="hidden
self.parser.add_argument("--embedding_size", type=int, default=300, he
self.parser.add_argument("--max_sub_l", type=int, default=300, help="max length for subtitle")
self.parser.add_argument("--max_vcpt_l", type=int, default=300, help="max length for visual concepts")
self.parser.add_argument("--max_vid_l", type=int, default=480, help="max length for video feature")
self.parser.add_argument("--vocab_size", type=int, default=0, help="vocabulary size")
self.parser.add_argument("--no_normalize_v", action="store_true", help="do not normalize video featrue")
```

The size of hidden state for the first LSTM

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# model config
self.parser.add_argument("--no_glove", action="store_true", help="not use glove vectors")
self.parser.add_argument("--no_ts", action="store_true", help="no timestep annotation, use full length feature")
self.parser.add_argument("--input_streams", type=str, nargs="+", choices=["vcpt", "sub", "imagenet"],
                         help="input streams for the model, will use both `vcpt` and `sub` streams")
self.parser.add_argument("--n_layers_cls", type=int, default=1, help="number of layers in classifier")
self.parser.add_argument("--hsz1", type=int, default=150, help="hidden size for the first lstm")
self.parser.add_argument("--hsz2", type=int, default=300, help="hidden size for the second lstm")
self.parser.add_argument("--embedding_size", type=int, default=300, he
self.parser.add_argument("--max_sub_l", type=int, default=300, help="ma
self.parser.add_argument("--max_vcpt_l", type=int, default=300, help="max length for visual concepts")
self.parser.add_argument("--max_vid_l", type=int, default=480, help="max length for video feature")
self.parser.add_argument("--vocab_size", type=int, default=0, help="vocabulary size")
self.parser.add_argument("--no_normalize_v", action="store_true", help="do not normalize video featrue")
```

The size of hidden state for the second LSTM

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# model config
self.parser.add_argument("--no_glove", action="store_true", help="not use glove vectors")
self.parser.add_argument("--no_ts", action="store_true", help="no timestep annotation, use full length feature")
self.parser.add_argument("--input_streams", type=str, nargs="+", choices=["vcpt", "sub", "imagenet"],
                         help="input streams for the model, will use both `vcpt` and `sub` streams")
self.parser.add_argument("--n_layers_cls", type=int, default=1, help="number of layers in classifier")
self.parser.add_argument("--hsz1", type=int, default=150, help="hidden size for the first lstm")
self.parser.add_argument("--hsz2", type=int, default=300, help="hidden size for the second lstm")
self.parser.add_argument("--embedding_size", type=int, default=300, help="word embedding dim")
self.parser.add_argument("--max_sub_l", type=int, default=300, help="ma
self.parser.add_argument("--max_vcpt_l", type=int, default=300, help="
self.parser.add_argument("--max_vid_l", type=int, default=480, help="max length for video feature")
self.parser.add_argument("--vocab_size", type=int, default=0, help="vocabulary size")
self.parser.add_argument("--no_normalize_v", action="store_true", help="do not normalize video featrue")
```

The dimension of word embedding

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# model config
self.parser.add_argument("--no_glove", action="store_true", help="not use glove vectors")
self.parser.add_argument("--no_ts", action="store_true", help="no timestep annotation, use full length feature")
self.parser.add_argument("--input_streams", type=str, nargs="+", choices=["vcpt", "sub", "imagenet"],
                         help="input streams for the model, will use both `vcpt` and `sub` streams")
self.parser.add_argument("--n_layers_cls", type=int, default=1, help="number of layers in classifier")
self.parser.add_argument("--hsz1", type=int, default=150, help="hidden size for the first lstm")
self.parser.add_argument("--hsz2", type=int, default=300, help="hidden
self.parser.add_argument("--embedding_size", type=int, default=300, help="word embedding dim")
self.parser.add_argument("--max_sub_l", type=int, default=300, help="max length for subtitle")
self.parser.add_argument("--max_vcpt_l", type=int, default=300, help="max length for visual concepts")
self.parser.add_argument("--max_vid_l", type=int, default=480, help="max length for video feature")
self.parser.add_argument("--vocab_size", type=int, default=0, help="vocabulary size")
self.parser.add_argument("--no_normalize_v", action="store_true", help="do not normalize video featrue")
```

Max length for each feature

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# path config
self.parser.add_argument("--train_path", type=str, default="./data/tvqa_train_processed.json",
                         help="train set path")
self.parser.add_argument("--valid_path", type=str, default="./data/tvqa_val_processed.json",
                         help="valid set path")
self.parser.add_argument("--test_path", type=str, default="./data/tvqa_test_public_processed.json",
                         help="test set path")
self.parser.add_argument("--glove_path", type=str, default="./data/glove.6B.300d.txt",
                         help="GloVe pretrained vector path")
self.parser.add_argument("--vcpt_path", type=str, default="./data/det_visual_concepts_hq.pickle",
                         help="visual concepts feature path")
self.parser.add_argument("--vid_feat_path", type=str, default="./data/tvqa_imagenet_pool5.h5",
                         help="imagenet feature path")
self.parser.add_argument("--vid_feat_size", type=int, default=2048,
                         help="visual feature dimension")
self.parser.add_argument("--word2idx_path", type=str, default="./cache/word2idx.pickle",
                         help="word2idx cache path")
self.parser.add_argument("--idx2word_path", type=str, default="./cache/idx2word.pickle",
                         help="idx2word cache path")
self.parser.add_argument("--vocab_embedding_path", type=str, default="./cache/vocab_embedding.pickle",
                         help="vocab_embedding cache path")
self.initialized = True
```

The paths of TVQA dataset

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# path config
self.parser.add_argument("--train_path", type=str, default="./data/tvqa_train_processed.json",
                         help="train set path")
self.parser.add_argument("--valid_path", type=str, default="./data/tvqa_val_processed.json",
                         help="valid set path")
self.parser.add_argument("--test_path", type=str, default="./data/tv                ocessed.json",
                         help="test set path")
self.parser.add_argument("--glove_path", type=str, default="./data/glove.6B.300d.txt",
                         help="GloVe pretrained vector path")
self.parser.add_argument("--vcpt_path", type=str, default="./data/det_visual_concepts_hq.pickle",
                         help="visual concepts feature path")
self.parser.add_argument("--vid_feat_path", type=str, default="./data/tvqa_imagenet_pool5.h5",
                         help="imagenet feature path")
self.parser.add_argument("--vid_feat_size", type=int, default=2048,
                         help="visual feature dimension")
self.parser.add_argument("--word2idx_path", type=str, default="./cache/word2idx.pickle",
                         help="word2idx cache path")
self.parser.add_argument("--idx2word_path", type=str, default="./cache/idx2word.pickle",
                         help="idx2word cache path")
self.parser.add_argument("--vocab_embedding_path", type=str, default="./cache/vocab_embedding.pickle",
                         help="vocab_embedding cache path")
self.initialized = True
```

The path of GloVe

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# path config
self.parser.add_argument("--train_path", type=str, default="./data/tvqa_train_processed.json",
                         help="train set path")
self.parser.add_argument("--valid_path", type=str, default="./data/tvqa_val_processed.json",
                         help="valid set path")
self.parser.add_argument("--test_path", type=str, default="./data/tvqa_test_public_processed.json",
                         help="test set path")
self.parser.add_argument("--glove_path", type=str, default="./data/glove.6B.300d.txt",
                         help="GloVe pretrained vector path")
self.parser.add_argument("--vcpt_path", type=str, default="./data/det_visual_concepts_hq.pickle",
                         help="visual concepts feature path")
self.parser.add_argument("--vid_feat_path", type=str, default="./data/tvqa_imagenet_pool5.h5",
                         help="imagenet feature path")
self.parser.add_argument("--vid_feat_size", type=int, default=2048,
                         help="visual feature dimension")
self.parser.add_argument("--word
                         help="w
self.parser.add_argument("--idx2word_path", type=str, default="./cache/idx2word.pickle",
                         help="idx2word cache path")
self.parser.add_argument("--vocab_embedding_path", type=str, default="./cache/vocab_embedding.pickle",
                         help="vocab_embedding cache path")
self.initialized = True
```

The paths and parameters of Visual concepts features and ImageNet features

https://github.com/jayleicn/TVQA

Argument Parser

>> vim config.py

```
# path config
self.parser.add_argument("--train_path", type=str, default="./data/tvqa_train_processed.json",
                         help="train set path")
self.parser.add_argument("--valid_path", type=str, default="./data/tvqa_val_processed.json",
                         help="valid set path")
self.parser.add_argument("--test_path", type=str, default="./data/tvqa_test_public_processed.json",
                         help="test set path")
self.parser.add_argument("--glove_path", type=str, default="./data/glove.6B.300d.txt",
                         help="GloVe pretrained vector path")
self.parser.add_argument("--vcpt_path", type=str, default="./data/det_visual_concepts_hq.pickle",
                         help="visual concepts feature path")
self.parser.add_argument("--vid_feat_path", type=str, default="./data/tvqa_imagenet_pool5.h5",
                         help="imagenet feature path")
self.parser.add_argument("--vid_feat_size", type=int, default=2048,
                         help="visual feature dimension")
self.parser.add_argument("--word2idx_path", type=str, default="./cache/word2idx.pickle",
                         help="word2idx cache path")
self.parser.add_argument("--idx2word_path", type=str, default="./cache/idx2word.pickle",
                         help="idx2word cache path")
self.parser.add_argument("--vocab_embedding_path", type=str, default="./cache/vocab_embedding.pickle",
                         help="vocab embedding cache path")
self.initialized = True
```

The paths of pre-processed files

https://github.com/jayleicn/TVQA

Model Initialization

>> vim model/tvqa_abc.py

```
class ABC(nn.Module):
    def __init__(self, opt):
        super(ABC, self).__init__()
        self.vid_flag = "imagenet" in opt.input_streams
        self.sub_flag = "sub" in opt.input_streams
        self.vcpt_flag = "vcpt" in opt.input_streams
        hidden_size_1 = opt.hsz1
        hidden_size_2 = opt.hsz2
        n_layers_cls = opt.n_layers_cls
        vid_feat_size = opt.vid_feat_size
        embedding_size = opt.embedding_size
        vocab_size = opt.vocab_size

        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.bidaf = BidafAttn(hidden_size_1 * 3, method="dot")  # no parameter for dot
        self.lstm_raw = RNNEncoder(300, hidden_size_1, bidirectional=True, dropout_p=0, n_layers=1, rnn_type="lstm")
```

The flags for whether using each feature or not

https://github.com/jayleicn/TVQA

Model Initialization

>> vim model/tvqa_abc.py

```
class ABC(nn.Module):
    def __init__(self, opt):
        super(ABC, self).__init__()
        self.vid_flag = "imagenet" in opt.input_streams
        self.sub_flag = "sub" in opt.input_streams
        self.vcpt_flag = "vcpt" in opt.input_streams
        hidden_size_1 = opt.hsz1
        hidden_size_2 = opt.hsz2
        n_layers_cls = opt.n_layers_cls
        vid_feat_size = opt.vid_feat_size
        embedding_size = opt.embedding_size
        vocab_size = opt.vocab_size

        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.bidaf = BidafAttn(hidden_size_1 * 3, method="dot")  # no parameter for dot
        self.lstm_raw = RNNEncoder(300, hidden_size_1, bidirectional=True, dropout_p=0, n_layers=1, rnn_type="lstm")
```

The parameters for the model

https://github.com/jayleicn/TVQA

Model Initialization

>> vim model/tvqa_abc.py

```
class ABC(nn.Module):
    def __init__(self, opt):
        super(ABC, self).__init__()
        self.vid_flag = "imagenet" in opt.input_streams
        self.sub_flag = "sub" in opt.input_streams
        self.vcpt_flag = "vcpt" in opt.input_streams
        hidden_size_1 = opt.hsz1
        hidden_size_2 = opt.hsz2
        n_layers_cls = opt.n_layers_cls
        vid_feat_size = opt.vid_feat_size
        embedding_size = opt.embedding_size
        vocab_size = opt.vocab_size

        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.bidaf = BidafAttn(hidden_size_1 * 3, method="dot")  # no parameter for dot
        self.lstm_raw = RNNEncoder(300, hidden_size_1, bidirectional=True, dropout_p=0, n_layers=1, rnn_type="lstm")
```

Word embedding layer

https://github.com/jayleicn/TVQA

Word Embedding

torch.nn.Embedding – 2 arguments

- 6 optional arguments
- 2 required arguments: ***num_embeddings, embedding_dim***

CLASS `torch.nn.Embedding(`*`num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None`*`)` [SOURCE]

Parameters

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector

https://pytorch.org/docs/stable/nn.html?highlight=nn%20embedding#torch.nn.Embedding

Model Initialization

>> vim model/tvqa_abc.py

```python
class ABC(nn.Module):
    def __init__(self, opt):
        super(ABC, self).__init__()
        self.vid_flag = "imagenet" in opt.input_streams
        self.sub_flag = "sub" in opt.input_streams
        self.vcpt_flag = "vcpt" in opt.input_streams
        hidden_size_1 = opt.hsz1
        hidden_size_2 = opt.hsz2
        n_layers_cls = opt.n_layers_cls
        vid_feat_size = opt.vid_feat_size
        embedding_size = opt.embedding_size
        vocab_size = opt.vocab_size

        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.bidaf = BidafAttn(hidden_size_1 * 3, method="dot")  # no parameter for dot
        self.lstm_raw = RNNEncoder(300, hidden_size_1, bidirectional=True, dropout_p=0, n_layers=1, rnn_type="lstm")
```

Attention layer (Context Matching)

https://github.com/jayleicn/TVQA

Model Initialization

>> vim model/tvqa_abc.py

```python
class ABC(nn.Module):
    def __init__(self, opt):
        super(ABC, self).__init__()
        self.vid_flag = "imagenet" in opt.input_streams
        self.sub_flag = "sub" in opt.input_streams
        self.vcpt_flag = "vcpt" in opt.input_streams
        hidden_size_1 = opt.hsz1
        hidden_size_2 = opt.hsz2
        n_layers_cls = opt.n_layers_cls
        vid_feat_size = opt.vid_feat_size
        embedding_size = opt.embedding_size
        vocab_size = opt.vocab_size

        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.bidaf = BidafAttn(hidden_size_1 * 3, method="dot")  # no parameter for dot
        self.lstm_raw = RNNEncoder(300, hidden_size_1, bidirectional=True, dropout_p=0, n_layers=1, rnn_type="lstm")
```

First LSTM layer

https://github.com/jayleicn/TVQA

Long Short-Term Memory

torch.nn.LSTM – 7 arguments

- 5 optional arguments
- 2 required arguments: ***input_size, hidden_size***

CLASS `torch.nn.LSTM(*args, **kwargs)` [SOURCE]

Parameters

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h

⋮

https://pytorch.org/docs/stable/nn.html?highlight=nn%20lstm#torch.nn.LSTM

Model Initialization

>> vim model/tvqa_abc.py

Layers for ImageNet Features

```
if self.vid_flag:
    print("activate video stream")
    self.video_fc = nn.Sequential(
        nn.Dropout(0.5),
        nn.Linear(vid_feat_size, embedding_size),
        nn.Tanh(),
    )
    self.lstm_mature_vid = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                      dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_vid = MLP(hidden_size_2*2, 1, 500, n_layers_cls)

if self.sub_flag:
    print("activate sub stream")
    self.lstm_mature_sub = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                      dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_sub = MLP(hidden_size_2*2, 1, 500, n_layers_cls)

if self.vcpt_flag:
    print("activate vcpt stream")
    self.lstm_mature_vcpt = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                       dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_vcpt = MLP(hidden_size_2*2, 1, 500, n_layers_cls)
```

https://github.com/jayleicn/TVQA

Model Initialization

>> vim model/tvqa_abc.py

```
if self.vid_flag:
    print("activate video stream")
    self.video_fc = nn.Sequential(
        nn.Dropout(0.5),
        nn.Linear(vid_feat_size, embedding_size),
        nn.Tanh(),
    )
    self.lstm_mature_vid = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                      dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_vid = MLP(hidden_size_2*2, 1, 500, n_layers_cls)

if self.sub_flag:
    print("activate sub stream")
    self.lstm_mature_sub = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                      dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_sub = MLP(hidden_size_2*2, 1, 500, n_layers_cls)

if self.vcpt_flag:
    print("activate vcpt stream")
    self.lstm_mature_vcpt = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                       dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_vcpt = MLP(hidden_size_2*2, 1, 500, n_layers_cls)
```

Layers for subtitle

https://github.com/jayleicn/TVQA

Model Initialization

>> vim model/tvqa_abc.py

```python
if self.vid_flag:
    print("activate video stream")
    self.video_fc = nn.Sequential(
        nn.Dropout(0.5),
        nn.Linear(vid_feat_size, embedding_size),
        nn.Tanh(),
    )
    self.lstm_mature_vid = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                      dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_vid = MLP(hidden_size_2*2, 1, 500, n_layers_cls)

if self.sub_flag:
    print("activate sub stream")
    self.lstm_mature_sub = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                      dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_sub = MLP(hidden_size_2*2, 1, 500, n_layers_cls)

if self.vcpt_flag:
    print("activate vcpt stream")
    self.lstm_mature_vcpt = RNNEncoder(hidden_size_1 * 2 * 5, hidden_size_2, bidirectional=True,
                                       dropout_p=0, n_layers=1, rnn_type="lstm")
    self.classifier_vcpt = MLP(hidden_size_2*2, 1, 500, n_layers_cls)
```

Layers for visual concepts features

https://github.com/jayleicn/TVQA

GloVe Initialization

>> vim model/tvqa_abc.py

```
def load_embedding(self, pretrained_embedding):
    self.embedding.weight.data.copy_(torch.from_numpy(pretrained_embedding))
```

https://github.com/jayleicn/TVQA

Inputs

>> vim model/tvqa_abc.py

>> (Line 55) import pdb; pdb.set_trace()

```
54     def forward(self, q, q_l, a0, a0_l, a1, a1_l, a2, a2_l, a3, a3_l, a4, a4_l,
55                 sub, sub_l, vcpt, vcpt_l, vid, vid_l):
56         import pdb; pdb.set_trace()
57         e_q = self.embedding(q)
58         e_a0 = self.embedding(a0)
59         e_a1 = self.embedding(a1)
60         e_a2 = self.embedding(a2)
61         e_a3 = self.embedding(a3)
62         e_a4 = self.embedding(a4)
```

https://github.com/jayleicn/TVQA

Inputs

>> python main.py --input_streams sub vcpt

```
def forward(self, q, q_l, a0, a0_l, a1, a1_l, a2, a2_l, a3, a3_l, a4, a4_l,
            sub, sub_l, vcpt, vcpt_l, vid, vid_l):
```

```
(Pdb) print(a2)
tensor([[735,    2]], device='cuda:0')
(Pdb)
```

```
(Pdb) print(a3)
tensor([[9197,    2]], device='cuda:0')
(Pdb)
```

```
(Pdb) print(a4)
tensor([[1387,    2]], device='cuda:0')
(Pdb)
```

```
(Pdb) print(vcpt)
tensor([[  139,   597,    64,  1398,   132,   597,  1832,   510,  1359,   122,
           139,   597,   315,   735,  1590,  1526,  1253,   611,  1398,  1590,
           132,  2505,  1396,   597,  1590,  1595,    58,  3274,   316,  1387,
           132,  2650,   597,   638,   980,  2972,  1590,  1117,   213,   597,
          1771,  1590,   418,  2972,   884,  2972,   132,  1398, 12328,  4302,
           510,     2]], device='cuda:0')
(Pdb)
```

https://github.com/jayleicn/TVQA

Embedding and Encoding Layer

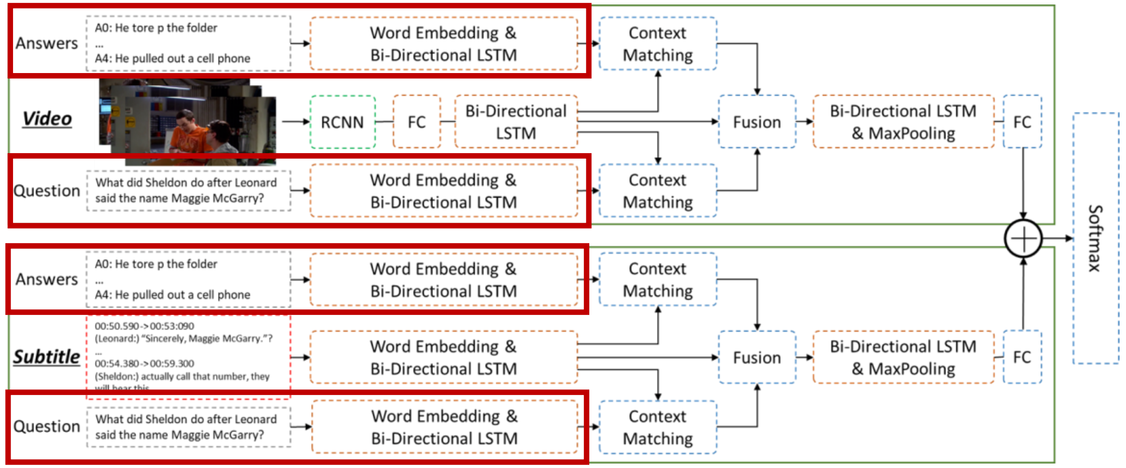
>> vim model/tvqa_abc.py or go to next (‘n’) using PDB

```
e_q = self.embedding(q)
e_a0 = self.embedding(a0)
e_a1 = self.embedding(a1)
e_a2 = self.embedding(a2)
e_a3 = self.embedding(a3)
e_a4 = self.embedding(a4)
```

Word embedding for question and answers

```
raw_out_q, _ = self.lstm_raw(e_q, q_l)
raw_out_a0, _ = self.lstm_raw(e_a0, a0_l)
raw_out_a1, _ = self.lstm_raw(e_a1, a1_l)
raw_out_a2, _ = self.lstm_raw(e_a2, a2_l)
raw_out_a3, _ = self.lstm_raw(e_a3, a3_l)
raw_out_a4, _ = self.lstm_raw(e_a4, a4_l)
```

Bi-directional LSTM for question and answers



The overview of Multi-Modal Video QA

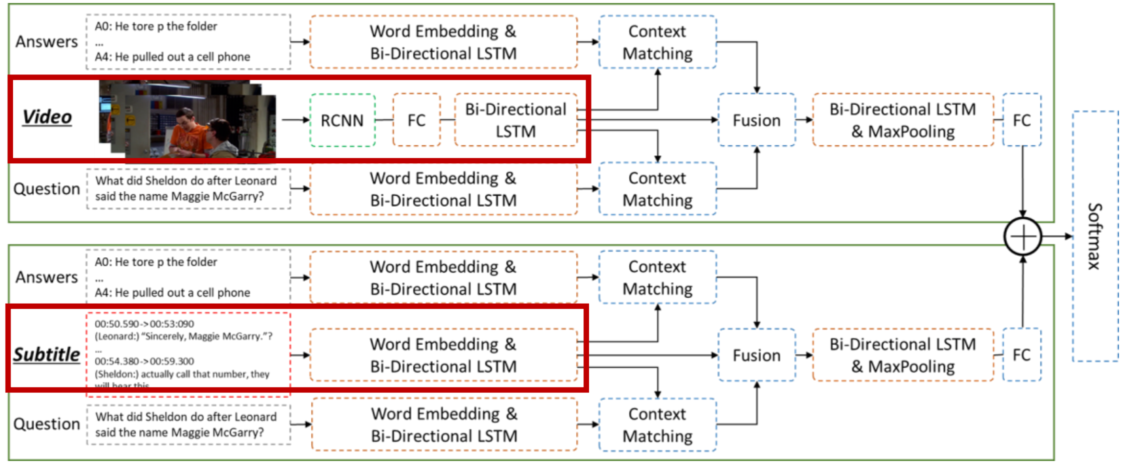
https://github.com/jayleicn/TVQA

Embedding and Encoding Layer

>> vim model/tvqa_abc.py or go to next (‘n’) using PDB

```python
if self.sub_flag:
    e_sub = self.embedding(sub)
    raw_out_sub, _ = self.lstm_raw(e_sub, sub_l)
    sub_out = self.stream_processor(self.lstm_mature_sub, self.classifier_sub, raw_out_sub, sub_l,
                                    raw_out_q, q_l, raw_out_a0, a0_l, raw_out_a1, a1_l,
                                    raw_out_a2, a2_l, raw_out_a3, a3_l, raw_out_a4, a4_l)
else:
    sub_out = 0

if self.vcpt_flag:
    e_vcpt = self.embedding(vcpt)
    raw_out_vcpt, _ = self.lstm_raw(e_vcpt, vcpt_l)
    vcpt_out = self.stream_processor(self.lstm_mature_vcpt, self.classifier_vcpt, raw_out_vcpt, vcpt_l,
                                     raw_out_q, q_l, raw_out_a0, a0_l, raw_out_a1, a1_l,
                                     raw_out_a2, a2_l, raw_out_a3, a3_l, raw_out_a4, a4_l)
else:
    vcpt_out = 0
```



The overview of Multi-Modal Video QA

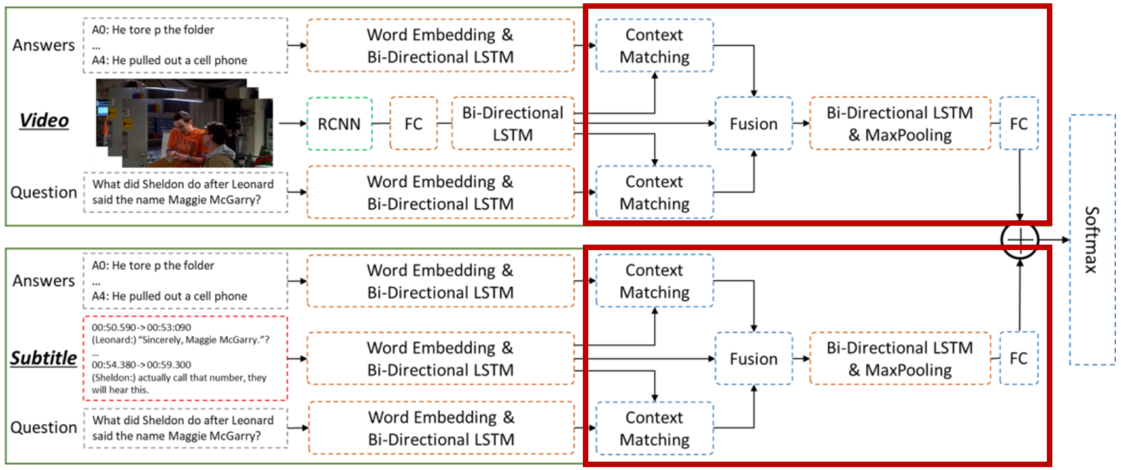
https://github.com/jayleicn/TVQA

Stream Processor

>> vim model/tvqa_abc.py or go to next ('n') using PDB

```
if self.sub_flag:
    e_sub = self.embedding(sub)
    raw_out_sub, _ = self.lstm_raw(e_sub, sub_l)
    sub_out = self.stream_processor(self.lstm_mature_sub, self.classifier_sub, raw_out_sub, sub_l,
                                    raw_out_q, q_l, raw_out_a0, a0_l, raw_out_a1, a1_l,
                                    raw_out_a2, a2_l, raw_out_a3, a3_l, raw_out_a4, a4_l)
else:
    sub_out = 0

if self.vcpt_flag:
    e_vcpt = self.embedding(vcpt)
    raw_out_vcpt, _ = self.lstm_raw(e_vcpt, vcpt_l)
    vcpt_out = self.stream_processor(self.lstm_mature_vcpt, self.classifier_vcpt, raw_out_vcpt, vcpt_l,
                                     raw_out_q, q_l, raw_out_a0, a0_l, raw_out_a1, a1_l,
                                     raw_out_a2, a2_l, raw_out_a3, a3_l, raw_out_a4, a4_l)
else:
    vcpt_out = 0
```



The overview of Multi-Modal Video QA

https://github.com/jayleicn/TVQA

Stream Processor

```
def stream_processor(self, lstm_mature, classifier, ctx_embed, ctx_l,
                     q_embed, q_l, a0_embed, a0_l, a1_embed, a1_l, a2_embed, a2_l, a3_embed, a3_l, a4_embed, a4_l):
    u_q, _ = self.bidaf(ctx_embed, ctx_l, q_embed, q_l)
    u_a0, _ = self.bidaf(ctx_embed, ctx_l, a0_embed, a0_l)
    u_a1, _ = self.bidaf(ctx_embed, ctx_l, a1_embed, a1_l)
    u_a2, _ = self.bidaf(ctx_embed, ctx_l, a2_embed, a2_l)
    u_a3, _ = self.bidaf(ctx_embed, ctx_l, a3_embed, a3_l)
    u_a4, _ = self.bidaf(ctx_embed, ctx_l, a4_embed, a4_l)

    concat_a0 = torch.cat([ctx_embed, u_a0, u_q, u_a0 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a1 = torch.cat([ctx_embed, u_a1, u_q, u_a1 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a2 = torch.cat([ctx_embed, u_a2, u_q, u_a2 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a3 = torch.cat([ctx_embed, u_a3, u_q, u_a3 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a4 = torch.cat([ctx_embed, u_a4, u_q, u_a4 * ctx_embed, u_q * ctx_embed], dim=-1)

    mature_maxout_a0, _ = lstm_mature(concat_a0, ctx_l)
    mature_maxout_a1, _ = lstm_mature(concat_a1, ctx_l)
    mature_maxout_a2, _ = lstm_mature(concat_a2, ctx_l)
    mature_maxout_a3, _ = lstm_mature(concat_a3, ctx_l)
    mature_maxout_a4, _ = lstm_mature(concat_a4, ctx_l)

    mature_maxout_a0 = max_along_time(mature_maxout_a0, ctx_l).unsqueeze(1)
    mature_maxout_a1 = max_along_time(mature_maxout_a1, ctx_l).unsqueeze(1)
    mature_maxout_a2 = max_along_time(mature_maxout_a2, ctx_l).unsqueeze(1)
    mature_maxout_a3 = max_along_time(mature_maxout_a3, ctx_l).unsqueeze(1)
    mature_maxout_a4 = max_along_time(mature_maxout_a4, ctx_l).unsqueeze(1)

    mature_answers = torch.cat([
        mature_maxout_a0, mature_maxout_a1, mature_maxout_a2, mature_maxout_a3, mature_maxout_a4
    ], dim=1)
    out = classifier(mature_answers)  # (B, 5)
    return out
```

Context matching

https://github.com/jayleicn/TVQA

Stream Processor

```python
def stream_processor(self, lstm_mature, classifier, ctx_embed, ctx_l,
                     q_embed, q_l, a0_embed, a0_l, a1_embed, a1_l, a2_embed, a2_l, a3_embed, a3_l, a4_embed, a4_l):
    u_q, _ = self.bidaf(ctx_embed, ctx_l, q_embed, q_l)
    u_a0, _ = self.bidaf(ctx_embed, ctx_l, a0_embed, a0_l)
    u_a1, _ = self.bidaf(ctx_embed, ctx_l, a1_embed, a1_l)
    u_a2, _ = self.bidaf(ctx_embed, ctx_l, a2_embed, a2_l)
    u_a3, _ = self.bidaf(ctx_embed, ctx_l, a3_embed, a3_l)
    u_a4, _ = self.bidaf(ctx_embed, ctx_l, a4_embed, a4_l)

    concat_a0 = torch.cat([ctx_embed, u_a0, u_q, u_a0 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a1 = torch.cat([ctx_embed, u_a1, u_q, u_a1 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a2 = torch.cat([ctx_embed, u_a2, u_q, u_a2 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a3 = torch.cat([ctx_embed, u_a3, u_q, u_a3 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a4 = torch.cat([ctx_embed, u_a4, u_q, u_a4 * ctx_embed, u_q * ctx_embed], dim=-1)

    mature_maxout_a0, _ = lstm_mature(concat_a0, ctx_l)
    mature_maxout_a1, _ = lstm_mature(concat_a1, ctx_l)
    mature_maxout_a2, _ = lstm_mature(concat_a2, ctx_l)
    mature_maxout_a3, _ = lstm_mature(concat_a3, ctx_l)
    mature_maxout_a4, _ = lstm_mature(concat_a4, ctx_l)

    mature_maxout_a0 = max_along_time(mature_maxout_a0, ctx_l).unsqueeze(1)
    mature_maxout_a1 = max_along_time(mature_maxout_a1, ctx_l).unsqueeze(1)
    mature_maxout_a2 = max_along_time(mature_maxout_a2, ctx_l).unsqueeze(1)
    mature_maxout_a3 = max_along_time(mature_maxout_a3, ctx_l).unsqueeze(1)
    mature_maxout_a4 = max_along_time(mature_maxout_a4, ctx_l).unsqueeze(1)

    mature_answers = torch.cat([
        mature_maxout_a0, mature_maxout_a1, mature_maxout_a2, mature_maxout_a3, mature_maxout_a4
    ], dim=1)
    out = classifier(mature_answers)  # (B, 5)
    return out
```

Fusion

https://github.com/jayleicn/TVQA

Stream Processor

```
def stream_processor(self, lstm_mature, classifier, ctx_embed, ctx_l,
                     q_embed, q_l, a0_embed, a0_l, a1_embed, a1_l, a2_embed, a2_l, a3_embed, a3_l, a4_embed, a4_l):
    u_q, _ = self.bidaf(ctx_embed, ctx_l, q_embed, q_l)
    u_a0, _ = self.bidaf(ctx_embed, ctx_l, a0_embed, a0_l)
    u_a1, _ = self.bidaf(ctx_embed, ctx_l, a1_embed, a1_l)
    u_a2, _ = self.bidaf(ctx_embed, ctx_l, a2_embed, a2_l)
    u_a3, _ = self.bidaf(ctx_embed, ctx_l, a3_embed, a3_l)
    u_a4, _ = self.bidaf(ctx_embed, ctx_l, a4_embed, a4_l)

    concat_a0 = torch.cat([ctx_embed, u_a0, u_q, u_a0 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a1 = torch.cat([ctx_embed, u_a1, u_q, u_a1 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a2 = torch.cat([ctx_embed, u_a2, u_q, u_a2 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a3 = torch.cat([ctx_embed, u_a3, u_q, u_a3 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a4 = torch.cat([ctx_embed, u_a4, u_q, u_a4 * ctx_embed, u_q * ctx_embed], dim=-1)

    mature_maxout_a0, _ = lstm_mature(concat_a0, ctx_l)
    mature_maxout_a1, _ = lstm_mature(concat_a1, ctx_l)
    mature_maxout_a2, _ = lstm_mature(concat_a2, ctx_l)
    mature_maxout_a3, _ = lstm_mature(concat_a3, ctx_l)
    mature_maxout_a4, _ = lstm_mature(concat_a4, ctx_l)

    mature_maxout_a0 = max_along_time(mature_maxout_a0, ctx_l).unsqueeze(1)
    mature_maxout_a1 = max_along_time(mature_maxout_a1, ctx_l).unsqueeze(1)
    mature_maxout_a2 = max_along_time(mature_maxout_a2, ctx_l).unsqueeze(1)
    mature_maxout_a3 = max_along_time(mature_maxout_a3, ctx_l).unsqueeze(1)
    mature_maxout_a4 = max_along_time(mature_maxout_a4, ctx_l).unsqueeze(1)

    mature_answers = torch.cat([
        mature_maxout_a0, mature_maxout_a1, mature_maxout_a2, mature_maxout_a3, mature_maxout_a4
    ], dim=1)
    out = classifier(mature_answers)  # (B, 5)
    return out
```

Bi-directional LSTM (Second LSTM)

https://github.com/jayleicn/TVQA

Stream Processor

```python
def stream_processor(self, lstm_mature, classifier, ctx_embed, ctx_l,
                     q_embed, q_l, a0_embed, a0_l, a1_embed, a1_l, a2_embed, a2_l, a3_embed, a3_l, a4_embed, a4_l):
    u_q, _ = self.bidaf(ctx_embed, ctx_l, q_embed, q_l)
    u_a0, _ = self.bidaf(ctx_embed, ctx_l, a0_embed, a0_l)
    u_a1, _ = self.bidaf(ctx_embed, ctx_l, a1_embed, a1_l)
    u_a2, _ = self.bidaf(ctx_embed, ctx_l, a2_embed, a2_l)
    u_a3, _ = self.bidaf(ctx_embed, ctx_l, a3_embed, a3_l)
    u_a4, _ = self.bidaf(ctx_embed, ctx_l, a4_embed, a4_l)

    concat_a0 = torch.cat([ctx_embed, u_a0, u_q, u_a0 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a1 = torch.cat([ctx_embed, u_a1, u_q, u_a1 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a2 = torch.cat([ctx_embed, u_a2, u_q, u_a2 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a3 = torch.cat([ctx_embed, u_a3, u_q, u_a3 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a4 = torch.cat([ctx_embed, u_a4, u_q, u_a4 * ctx_embed, u_q * ctx_embed], dim=-1)

    mature_maxout_a0, _ = lstm_mature(concat_a0, ctx_l)
    mature_maxout_a1, _ = lstm_mature(concat_a1, ctx_l)
    mature_maxout_a2, _ = lstm_mature(concat_a2, ctx_l)
    mature_maxout_a3, _ = lstm_mature(concat_a3, ctx_l)
    mature_maxout_a4, _ = lstm_mature(concat_a4, ctx_l)

    mature_maxout_a0 = max_along_time(mature_maxout_a0, ctx_l).unsqueeze(1)
    mature_maxout_a1 = max_along_time(mature_maxout_a1, ctx_l).unsqueeze(1)
    mature_maxout_a2 = max_along_time(mature_maxout_a2, ctx_l).unsqueeze(1)
    mature_maxout_a3 = max_along_time(mature_maxout_a3, ctx_l).unsqueeze(1)
    mature_maxout_a4 = max_along_time(mature_maxout_a4, ctx_l).unsqueeze(1)

    mature_answers = torch.cat([
        mature_maxout_a0, mature_maxout_a1, mature_maxout_a2, mature_maxout_a3, mature_maxout_a4
    ], dim=1)
    out = classifier(mature_answers)  # (B, 5)
    return out
```

Max pooling

https://github.com/jayleicn/TVQA

Stream Processor

```
def stream_processor(self, lstm_mature, classifier, ctx_embed, ctx_l,
                     q_embed, q_l, a0_embed, a0_l, a1_embed, a1_l, a2_embed, a2_l, a3_embed, a3_l, a4_embed, a4_l):
    u_q, _ = self.bidaf(ctx_embed, ctx_l, q_embed, q_l)
    u_a0, _ = self.bidaf(ctx_embed, ctx_l, a0_embed, a0_l)
    u_a1, _ = self.bidaf(ctx_embed, ctx_l, a1_embed, a1_l)
    u_a2, _ = self.bidaf(ctx_embed, ctx_l, a2_embed, a2_l)
    u_a3, _ = self.bidaf(ctx_embed, ctx_l, a3_embed, a3_l)
    u_a4, _ = self.bidaf(ctx_embed, ctx_l, a4_embed, a4_l)

    concat_a0 = torch.cat([ctx_embed, u_a0, u_q, u_a0 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a1 = torch.cat([ctx_embed, u_a1, u_q, u_a1 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a2 = torch.cat([ctx_embed, u_a2, u_q, u_a2 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a3 = torch.cat([ctx_embed, u_a3, u_q, u_a3 * ctx_embed, u_q * ctx_embed], dim=-1)
    concat_a4 = torch.cat([ctx_embed, u_a4, u_q, u_a4 * ctx_embed, u_q * ctx_embed], dim=-1)

    mature_maxout_a0, _ = lstm_mature(concat_a0, ctx_l)
    mature_maxout_a1, _ = lstm_mature(concat_a1, ctx_l)
    mature_maxout_a2, _ = lstm_mature(concat_a2, ctx_l)
    mature_maxout_a3, _ = lstm_mature(concat_a3, ctx_l)
    mature_maxout_a4, _ = lstm_mature(concat_a4, ctx_l)

    mature_maxout_a0 = max_along_time(mature_maxout_a0, ctx_l).unsqueeze(1)
    mature_maxout_a1 = max_along_time(mature_maxout_a1, ctx_l).unsqueeze(1)
    mature_maxout_a2 = max_along_time(mature_maxout_a2, ctx_l).unsqueeze(1)
    mature_maxout_a3 = max_along_time(mature_maxout_a3, ctx_l).unsqueeze(1)
    mature_maxout_a4 = max_along_time(mature_maxout_a4, ctx_l).unsqueeze(1)

    mature_answers = torch.cat([
        mature_maxout_a0, mature_maxout_a1, mature_maxout_a2, mature_maxout_a3, mature_maxout_a4
    ], dim=1)
    out = classifier(mature_answers)  # (B, 5)
    return out
```

Fusion and FC

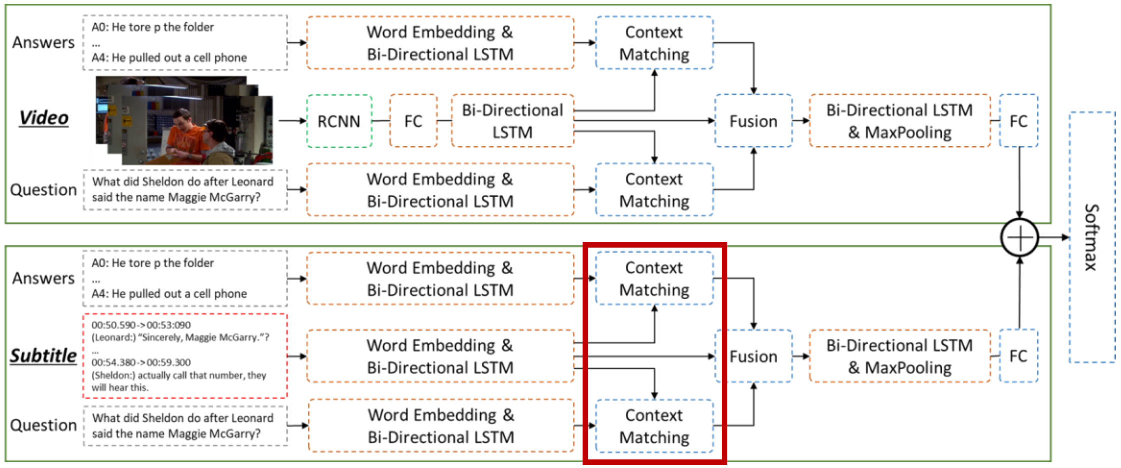
https://github.com/jayleicn/TVQA

Context Matching

>> vim model/tvqa_abc.py or go to step into ('s') using PDB

```
u_q, _ = self.bidaf(ctx_embed, ctx_l, q_embed, q_l)
u_a0, _ = self.bidaf(ctx_embed, ctx_l, a0_embed, a0_l)
u_a1, _ = self.bidaf(ctx_embed, ctx_l, a1_embed, a1_l)
u_a2, _ = self.bidaf(ctx_embed, ctx_l, a2_embed, a2_l)
u_a3, _ = self.bidaf(ctx_embed, ctx_l, a3_embed, a3_l)
u_a4, _ = self.bidaf(ctx_embed, ctx_l, a4_embed, a4_l)
```

Calculation for attention value from subtitle by question and answers



The overview of Multi-Modal Video QA

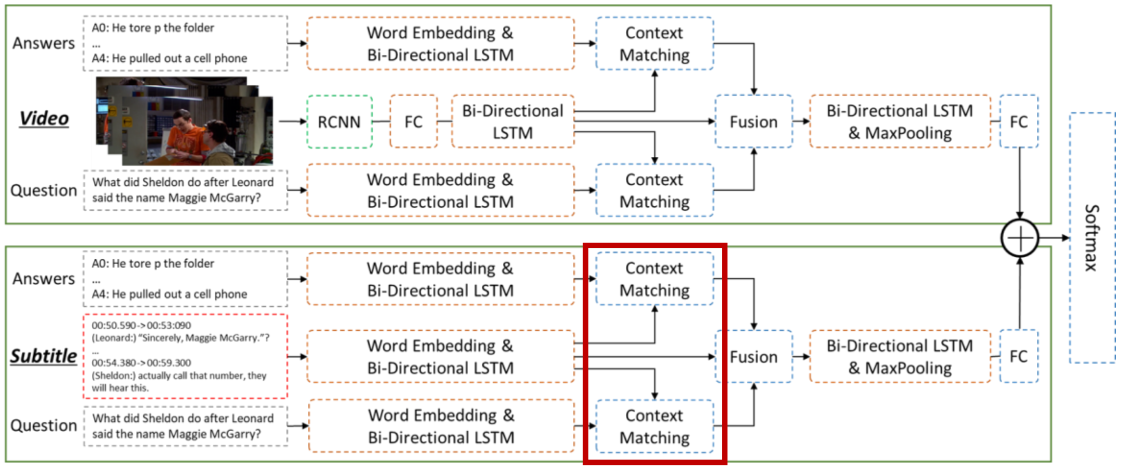
https://github.com/jayleicn/TVQA

Context Matching

>> vim model/bidaf.py or go to step into ('s') using PDB

```python
def forward(self, s1, l1, s2, l2):
    s = self.similarity(s1, l1, s2, l2)
    u_tile = self.get_u_tile(s, s2)
    # h_tile = self.get_h_tile(s, s1)
    h_tile = self.get_h_tile(s, s1) if self.get_h else None
    return u_tile, h_tile
    # return u_tile
```

Attention score



The overview of Multi-Modal Video QA

https://github.com/jayleicn/TVQA

Context Matching

>> vim model/bidaf.py or go to step into ('s') using PDB

```
def similarity(self, s1, l1, s2, l2):
    """
    :param s1: [B, t1, D]
    :param l1: [B]
    :param s2: [B, t2, D]
    :param l2: [B]
    :return:
    """
    if self.method == "original":
        t1 = s1.size(1)
        t2 = s2.size(1)
        repeat_s1 = s1.unsqueeze(2).repeat(1, 1, t2, 1)  # [B, T1, T2, D]
        repeat_s2 = s2.unsqueeze(1).repeat(1, t1, 1, 1)  # [B, T1, T2, D]
        packed_s1_s2 = torch.cat([repeat_s1, repeat_s2, repeat_s1 * repeat_s2], dim=3)  # [B, T1, T2, D*3]
        s = self.mlp(packed_s1_s2).squeeze()  # s is the similarity matrix from biDAF paper. [B, T1, T2]
    elif self.method == "dot":
        s = torch.bmm(s1, s2.transpose(1, 2))

    s_mask = s.data.new(*s.size()).fill_(1).byte()  # [B, T1, T2]
    # Init similarity mask using lengths
    for i, (l_1, l_2) in enumerate(zip(l1, l2)):
        s_mask[i][:l_1, :l_2] = 0

    s_mask = Variable(s_mask)
    s.data.masked_fill_(s_mask.data.byte(), -float("inf"))
    return s
```

Attention score

https://github.com/jayleicn/TVQA

Context Matching

>> vim model/bidaf.py or go to step into ('s') using PDB

```
def similarity(self, s1, l1, s2, l2):
    """
    :param s1: [B, t1, D]
    :param l1: [B]
    :param s2: [B, t2, D]
    :param l2: [B]
    :return:
    """
    if self.method == "original":
        t1 = s1.size(1)
        t2 = s2.size(1)
        repeat_s1 = s1.unsqueeze(2).repeat(1, 1, t2, 1)  # [B, T1, T2, D]
        repeat_s2 = s2.unsqueeze(1).repeat(1, t1, 1, 1)  # [B, T1, T2, D]
        packed_s1_s2 = torch.cat([repeat_s1, repeat_s2, repeat_s1 * repeat_s2], dim=3)  # [B, T1, T2, D*3]
        s = self.mlp(packed_s1_s2).squeeze()  # s is the similarity matrix from biDAF paper. [B, T1, T2]
    elif self.method == "dot":
        s = torch.bmm(s1, s2.transpose(1, 2))

    s_mask = s.data.new(*s.size()).fill_(1).byte()  # [B, T1, T2]
    # Init similarity mask using lengths
    for i, (l_1, l_2) in enumerate(zip(l1, l2)):
        s_mask[i][:l_1, :l_2] = 0

    s_mask = Variable(s_mask)
    s.data.masked_fill_(s_mask.data.byte(), -float("inf"))
    return s
```

Bi-directional attention score

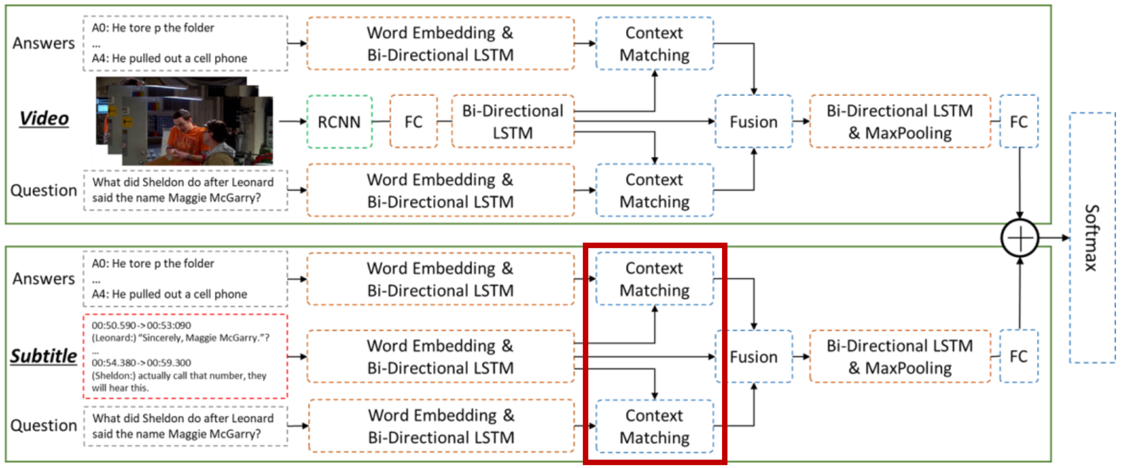
https://github.com/jayleicn/TVQA

Context Matching

>> vim model/bidaf.py or go to step into ('s') using PDB

```
def forward(self, s1, l1, s2, l2):
    s = self.similarity(s1, l1, s2, l2)
    u_tile = self.get_u_tile(s, s2)
    # h_tile = self.get_h_tile(s, s1)
    h_tile = self.get_h_tile(s, s1) if self.get_h else None
    return u_tile, h_tile
    # return u_tile
```

Attention value



The overview of Multi-Modal Video QA

https://github.com/jayleicn/TVQA

Context Matching

>> vim model/bidaf.py or go to step into ('s') using PDB

```
@classmethod
def get_u_tile(cls, s, s2):
    """
    attended vectors of s2 for each word in s1,
    signify which words in s2 are most relevant to words in s1
    """
    a_weight = F.softmax(s, dim=2)  # [B, t1, t2]
    a_weight.data.masked_fill_(a_weight.data != a_weight.data, 0)  # remove nan from softmax on -inf
    u_tile = torch.bmm(a_weight, s2)  # [B, t1, t2] * [B, t2, D] -> [B, t1, D]
    return u_tile
```

Attention weight

https://github.com/jayleicn/TVQA

Context Matching

>> vim model/bidaf.py or go to step into ('s') using PDB

```
@classmethod
def get_u_tile(cls, s, s2):
    """
    attended vectors of s2 for each word in s1,
    signify which words in s2 are most relevant to words in s1
    """
    a_weight = F.softmax(s, dim=2)  # [B, t1, t2]
    a_weight.data.masked_fill_(a_weight.data != a_weight.data, 0)  # remove nan from softmax on -inf
    u_tile = torch.bmm(a_weight, s2)  # [B, t1, t2] * [B, t2, D] -> [B, t1, D]
    return u_tile
```

Attention value

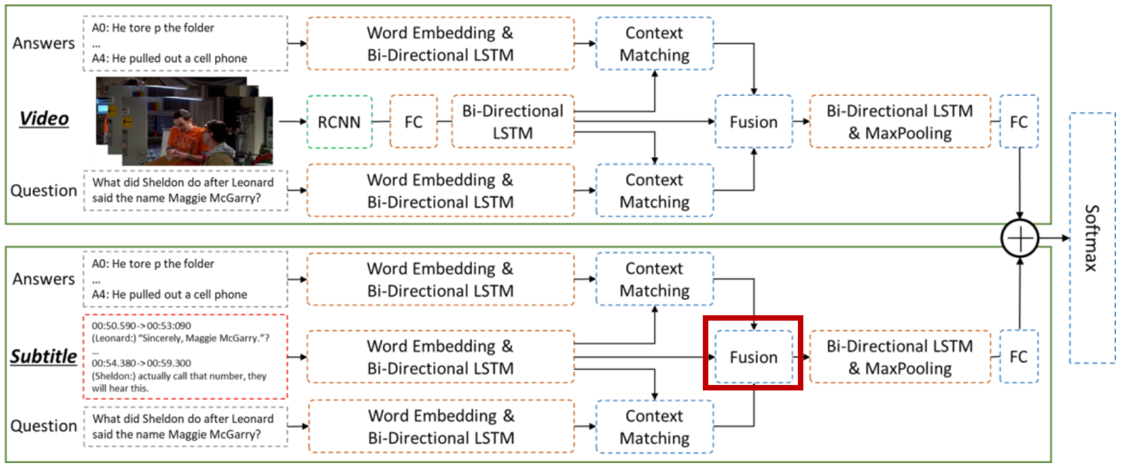
https://github.com/jayleicn/TVQA

Fusion

>> vim model/tvqa_abc.py or go to next (‘n’) using PDB

```
concat_a0 = torch.cat([ctx_embed, u_a0, u_q, u_a0 * ctx_embed, u_q * ctx_embed], dim=-1)
concat_a1 = torch.cat([ctx_embed, u_a1, u_q, u_a1 * ctx_embed, u_q * ctx_embed], dim=-1)
concat_a2 = torch.cat([ctx_embed, u_a2, u_q, u_a2 * ctx_embed, u_q * ctx_embed], dim=-1)
concat_a3 = torch.cat([ctx_embed, u_a3, u_q, u_a3 * ctx_embed, u_q * ctx_embed], dim=-1)
concat_a4 = torch.cat([ctx_embed, u_a4, u_q, u_a4 * ctx_embed, u_q * ctx_embed], dim=-1)
```

Concatenate all representations to one



The overview of Multi-Modal Video QA

https://github.com/jayleicn/TVQA

Fusion

torch.cat – 3 arguments

- 2 optional arguments
- 1 required arguments: ***tensors***

```
torch.cat(tensors, dim=0, out=None) → Tensor
```

Parameters

- **tensors** (*sequence of Tensors*) – any python sequence of tensors of the same type. Non-empty tensors provided must have the same shape, except in the cat dimension.
- **dim** (*int, optional*) – the dimension over which the tensors are concatenated
- **out** (*Tensor, optional*) – the output tensor

```
concat_a0 = torch.cat([ctx_embed, u_a0, u_q, u_a0 * ctx_embed, u_q * ctx_embed], dim=-1)
concat_a1 = torch.cat([ctx_embed, u_a1, u_q, u_a1 * ctx_embed, u_q * ctx_embed], dim=-1)
concat_a2 = torch.cat([ctx_embed, u_a2, u_q, u_a2 * ctx_embed, u_q * ctx_embed], dim=-1)
concat_a3 = torch.cat([ctx_embed, u_a3, u_q, u_a3 * ctx_embed, u_q * ctx_embed], dim=-1)
concat_a4 = torch.cat([ctx_embed, u_a4, u_q, u_a4 * ctx_embed, u_q * ctx_embed], dim=-1)
```

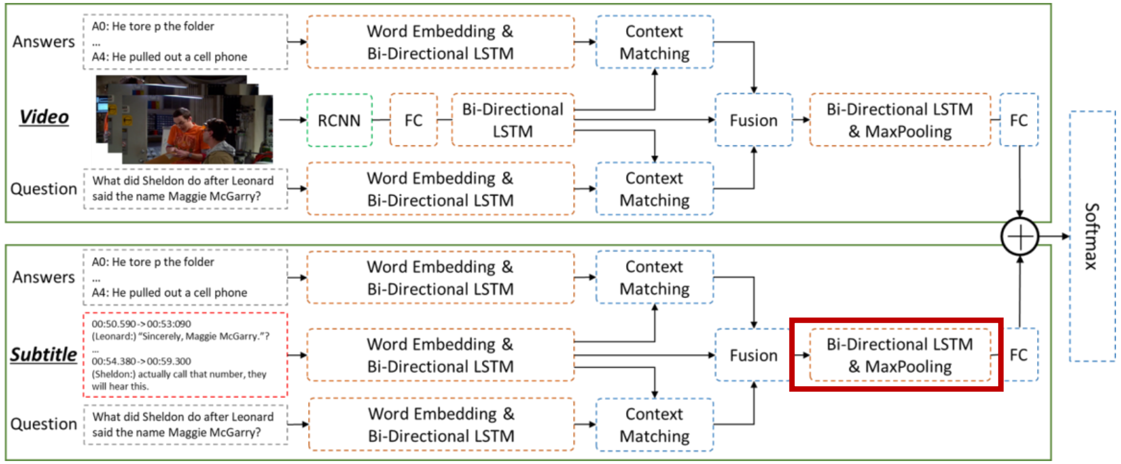
https://pytorch.org/docs/stable/nn.html?highlight=nn%20lstm#torch.cat

Bi-directional LSTM

>> vim model/tvqa_abc.py or go to next ('n') using PDB

```
mature_maxout_a0, _ = lstm_mature(concat_a0, ctx_l)
mature_maxout_a1, _ = lstm_mature(concat_a1, ctx_l)
mature_maxout_a2, _ = lstm_mature(concat_a2, ctx_l)
mature_maxout_a3, _ = lstm_mature(concat_a3, ctx_l)
mature_maxout_a4, _ = lstm_mature(concat_a4, ctx_l)
```

Second LSTM for question and answers



The overview of Multi-Modal Video QA

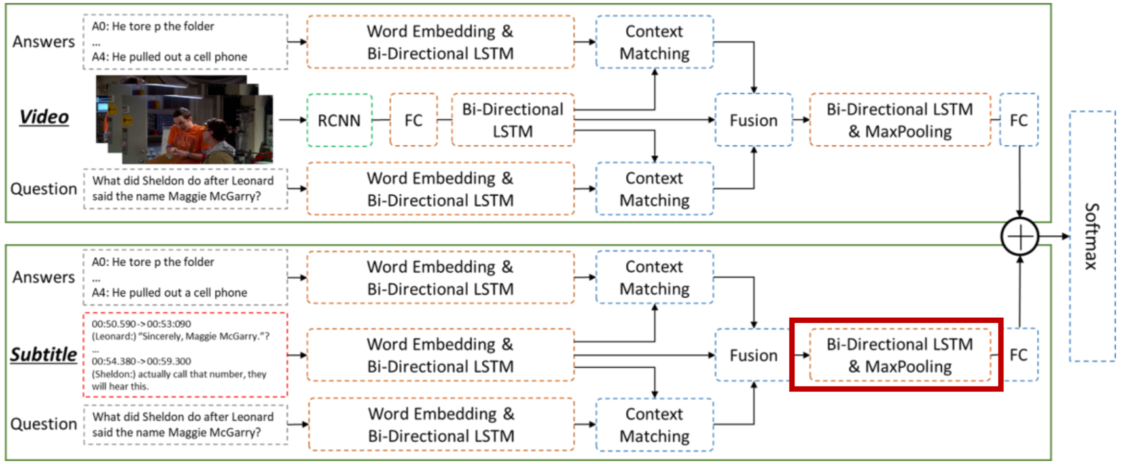
https://github.com/jayleicn/TVQA

Max Pooling

>> vim model/tvqa_abc.py or go to next (‘n’) using PDB

```
mature_maxout_a0 = max_along_time(mature_maxout_a0, ctx_l).unsqueeze(1)
mature_maxout_a1 = max_along_time(mature_maxout_a1, ctx_l).unsqueeze(1)
mature_maxout_a2 = max_along_time(mature_maxout_a2, ctx_l).unsqueeze(1)
mature_maxout_a3 = max_along_time(mature_maxout_a3, ctx_l).unsqueeze(1)
mature_maxout_a4 = max_along_time(mature_maxout_a4, ctx_l).unsqueeze(1)
```

Max pool along with time axis



The overview of Multi-Modal Video QA

https://github.com/jayleicn/TVQA

Max Pooling

max_along_time – 2 arguments

- 2 required arguments: ***outputs, lengths***

```
def max_along_time(outputs, lengths):
    """ Get maximum responses from RNN outputs along time axis
    :param outputs: (B, T, D)
    :param lengths: (B, )
    :return: (B, D)
    """
    outputs = [outputs[i, :int(lengths[i]), :].max(dim=0)[0] for i in range(len(lengths))]
    return torch.stack(outputs, dim=0)
```

```
        mature_maxout_a0 = max_along_time(mature_maxout_a0, ctx_l).unsqueeze(1)
        mature_maxout_a1 = max_along_time(mature_maxout_a1, ctx_l).unsqueeze(1)
        mature_maxout_a2 = max_along_time(mature_maxout_a2, ctx_l).unsqueeze(1)
        mature_maxout_a3 = max_along_time(mature_maxout_a3, ctx_l).unsqueeze(1)
        mature_maxout_a4 = max_along_time(mature_maxout_a4, ctx_l).unsqueeze(1)
```

Max Pooling

torch.max – 1 arguments

- 1 required arguments: ***tensors***

`torch.max(input)` → Tensor

Returns the maximum value of all elements in the `input` tensor.

Parameters

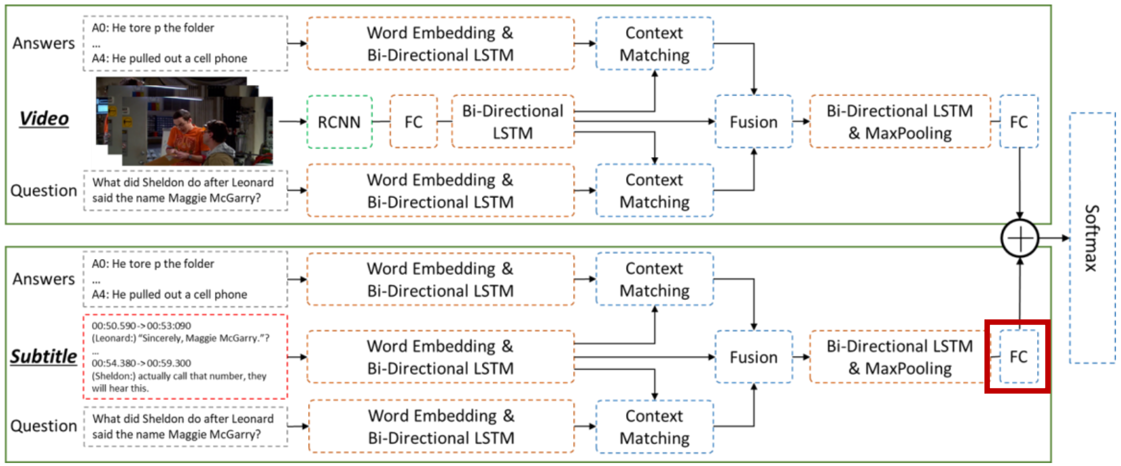
input (*Tensor*) – the input tensor

https://pytorch.org/docs/stable/nn.html?highlight=nn%20lstm#torch.max

FC

>> vim model/tvqa_abc.py or go to next ('n') using PDB

```
mature_answers = torch.cat([
    mature_maxout_a0, mature_maxout_a1, mature_maxout_a2, mature_maxout_a3, mature_maxout_a4
], dim=1)
out = classifier(mature_answers)  # (B, 5)
return out
```



The overview of Multi-Modal Video QA

https://github.com/jayleicn/TVQA

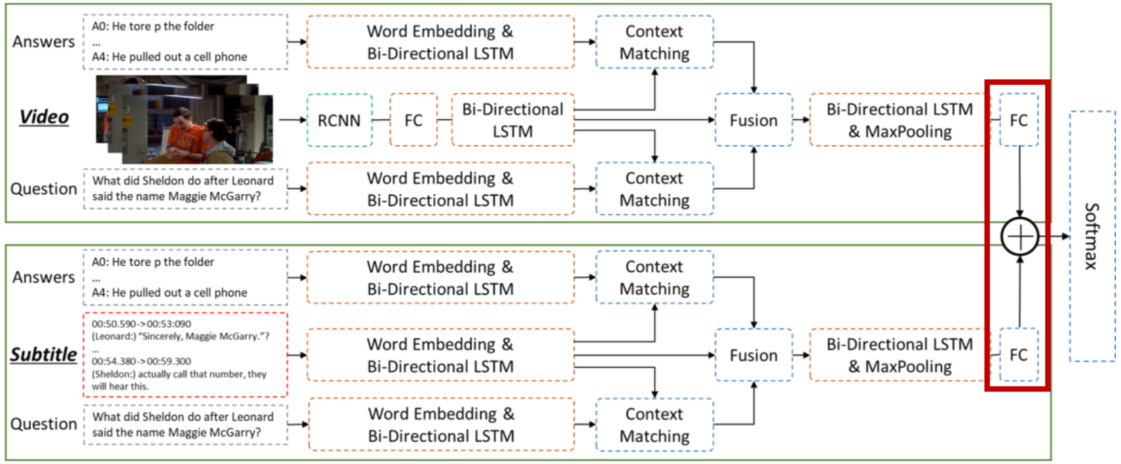
Prediction

>> vim model/tvqa_abc.py or go to next ('n') using PDB

```python
if self.sub_flag:
    e_sub = self.embedding(sub)
    raw_out_sub, _ = self.lstm_raw(e_sub, sub_l)
    sub_out = self.stream_processor(self.lstm_mature_sub, self.classifier_sub, raw_out_sub, sub_l,
                                    raw_out_q, q_l, raw_out_a0, a0_l, raw_out_a1, a1_l,
                                    raw_out_a2, a2_l, raw_out_a3, a3_l, raw_out_a4, a4_l)
else:
    sub_out = 0

if self.vcpt_flag:
    e_vcpt = self.embedding(vcpt)
    raw_out_vcpt, _ = self.lstm_raw(e_vcpt, vcpt_l)
    vcpt_out = self.stream_processor(self.lstm_mature_vcpt, self.classifier_vcpt, raw_out_vcpt, vcpt_l,
                                     raw_out_q, q_l, raw_out_a0, a0_l, raw_out_a1, a1_l,
                                     raw_out_a2, a2_l, raw_out_a3, a3_l, raw_out_a4, a4_l)
else:
    vcpt_out = 0
```

```python
out = sub_out + vcpt_out + vid_out  # adding zeros has no effect on backward
return out.squeeze()
```



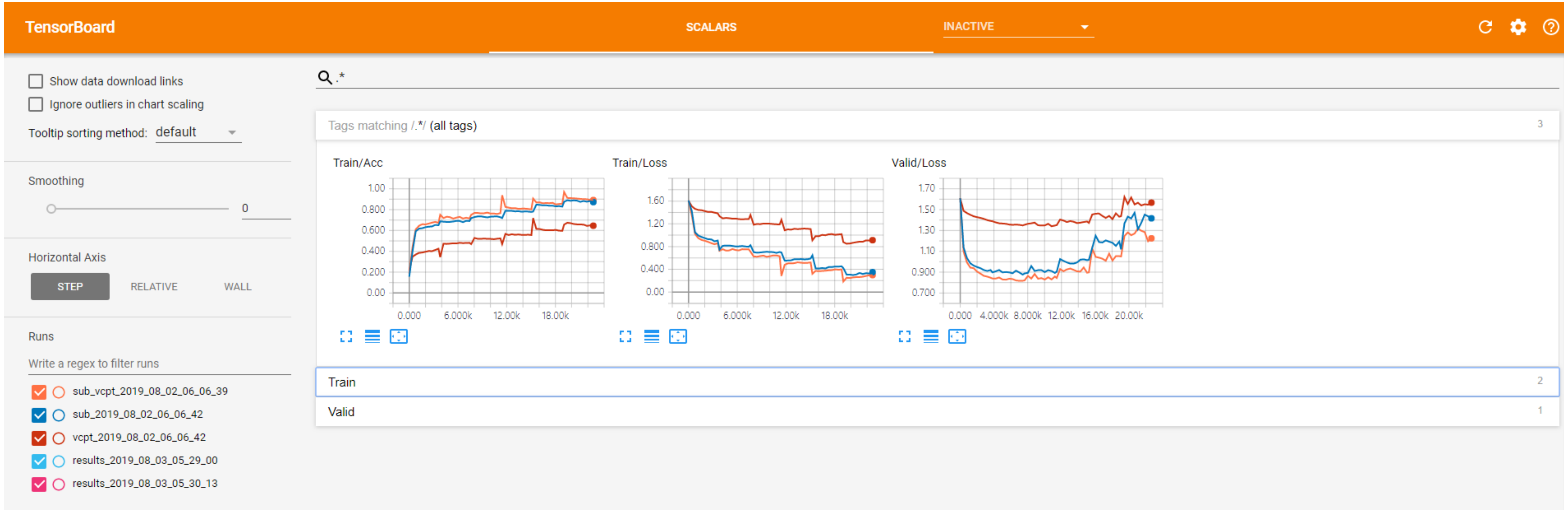
The overview of Multi-Modal Video QA

https://github.com/jayleicn/TVQA

Visualization of Training Process

>> tensorboard --logdir [results_dir] --port [Number]

```
mshan@ai8:/st2/mshan/models/ex2/TVQA$ tensorboard --logdir=results --port=5005
TensorBoard 0.1.8 at http://ai8:5005 (Press CTRL+C to quit)
```



https://github.com/jayleicn/TVQA

Inference

>> python test.py --model_dir [results_dir] --mode valid

```
mshan@ai8:/st2/mshan/models/ex2/TVQA$ python test.py --model_dir sub_vcpt_2019_08_02_06_06_39 --mode valid

Loading cache ...
activate sub stream
activate vcpt stream
153it [01:59,  1.44it/s]
In valid mode, accuracy is 0.6909
mshan@ai8:/st2/mshan/models/ex2/TVQA$
```

https://github.com/jayleicn/TVQA

Assignments

We have 2 assignments for Multi-Modal Video QA on TVQA dataset.

1. Why did TDIDF model in TVQA paper show good performance? (2.5 points)
   (See https://arxiv.org/abs/1809.01696)

2. What is the main difference between TVQA and TVQA+? (In terms of model) (2.5 points)
   (See https://arxiv.org/abs/1809.01696 and https://arxiv.org/abs/1904.11574)

Submission to ***mshan92@kaist.ac.kr***

Any questions?