Question Answering + Visual Question Answering实践环节

玖强

Story Comprehension

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk. Joe travelled to his office. Joe left the milk. Joe went to the bathroom.

Questions from Joe's angry mother:

Q1 : Where is Joe?

Q2 : Where is the milk now?

Q3 : Where was Joe before the office?

Dynamic Memory Networks

Extends simple RNNs with an *iterative attention mechanism* that focuses on one fact at a time and enables transitive reasoning

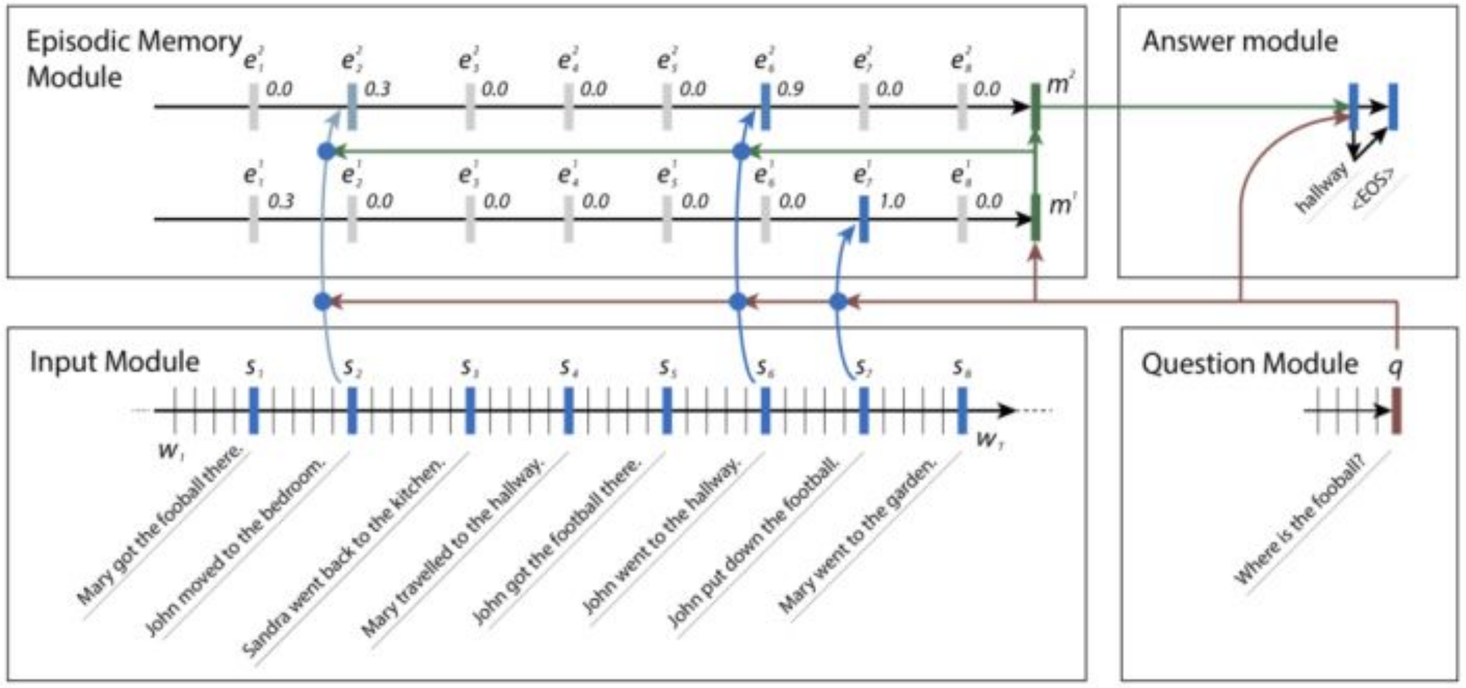
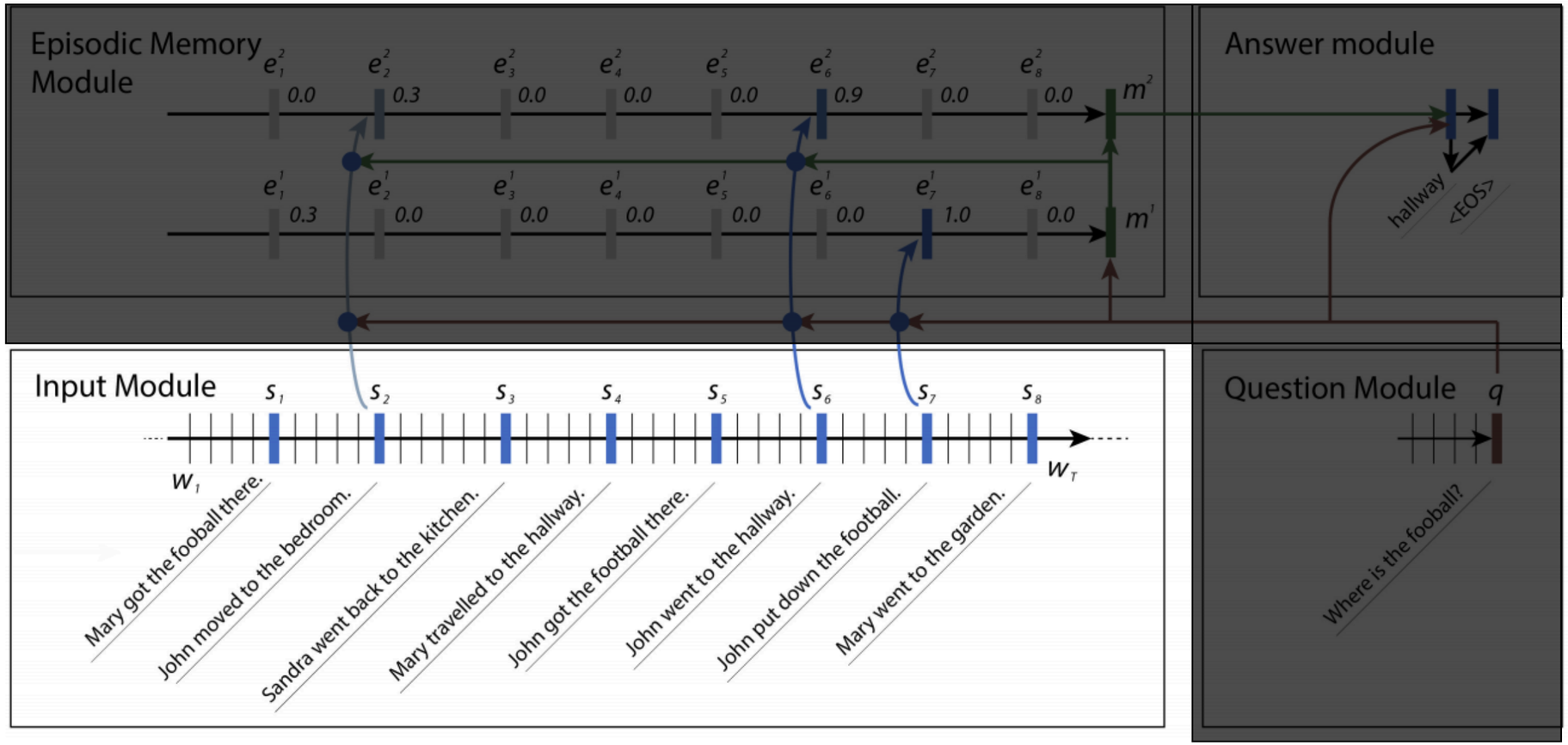


Figure 3. Real example of an input list of sentences and the attention gates that are triggered by a specific question from the bAbI tasks (Weston et al., 2015a). Gate values g_t^i are shown above the corresponding vectors. The gates change with each search over inputs. We do not draw connections for gates that are close to zero. Note that the second iteration has wrongly placed some weight in sentence 2, which makes some intuitive sense, as sentence 2 is another place John had been.

Ask Me Even More: Dynamic Memory Tensor Networks (Extended Model)

DMN-输入模块



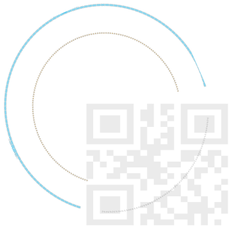
Final GRU Output for t^{th} sentence

$$c_t = \mathrm{GRU}(w_t^i, c_t^{i-1})$$

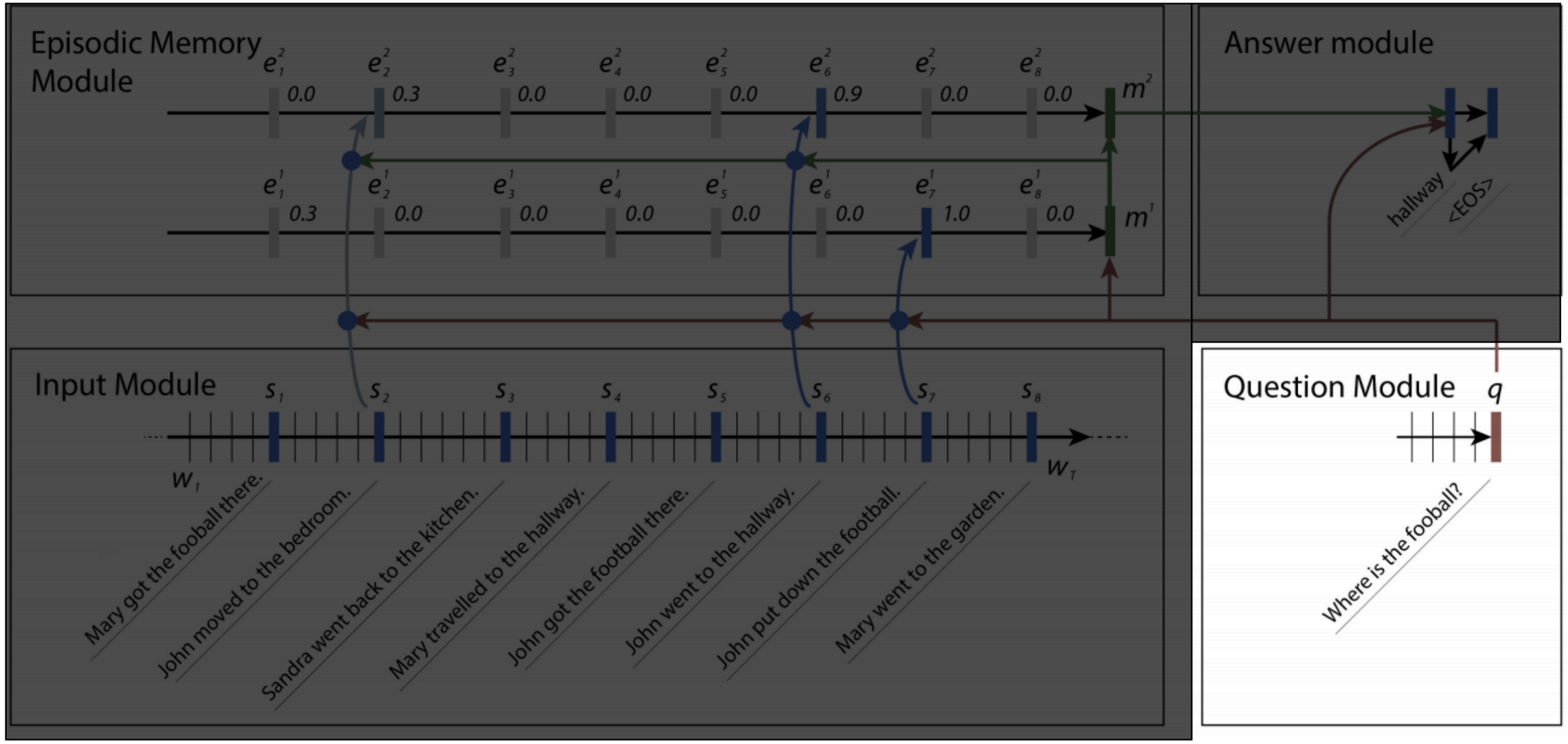
Ask Me Anything: Dynamic Memory Networks for Natural Language Processing, Kumar et. al. ICML 2016

DMN-输入模块

```
class InputModule(nn.Module):
    def __init__(self, vocab_size, hidden_size):
        super(InputModule, self).__init__()
        self.hidden_size = hidden_size
        self.gru = nn.GRU(hidden_size, hidden_size, bidirectional=True, batch_first=True)
        for name, param in self.gru.state_dict().items():
            if 'weight' in name: init.xavier_normal(param)
        self.dropout = nn.Dropout(0.1)

    def forward(self, contexts, word_embedding):
        '''
        contexts.size() -> (#batch, #sentence, #token)
        word_embedding() -> (#batch, #sentence x #token, #embedding)
        position_encoding() -> (#batch, #sentence, #embedding)
        facts.size() -> (#batch, #sentence, #hidden = #embedding)
        '''
        batch_num, sen_num, token_num = contexts.size()

        contexts = contexts.view(batch_num, -1)
        contexts = word_embedding(contexts)

        contexts = contexts.view(batch_num, sen_num, token_num, -1)
        contexts = position_encoding(contexts)
        contexts = self.dropout(contexts)

        h0 = Variable(torch.zeros(2, batch_num, self.hidden_size).cuda())
        facts, hdn = self.gru(contexts, h0)
        facts = facts[:, :, :hidden_size] + facts[:, :, hidden_size:]
        return facts
```



DMN-问题模块

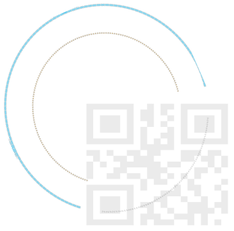


$$q = \mathrm{GRU}(q_w^i, q^{i-1})$$

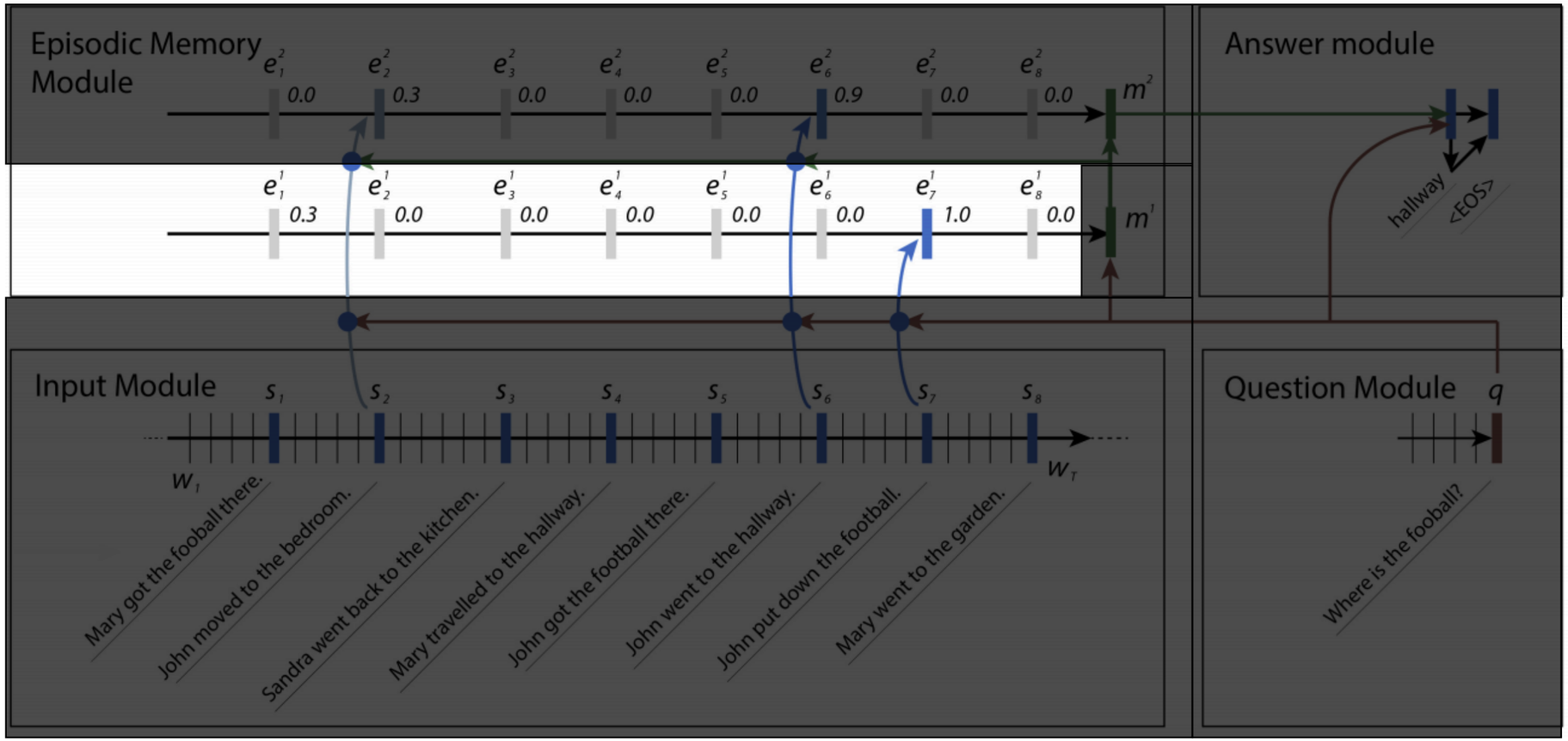
Ask Me Anything: Dynamic Memory Networks for Natural Language Processing, Kumar et. al. ICML 2016

DMN-问题模块

```
class QuestionModule(nn.Module):
    def __init__(self, vocab_size, hidden_size):
        super(QuestionModule, self).__init__()
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)

    def forward(self, questions, word_embedding):
        '''
        questions.size() -> (#batch, #token)
        word_embedding() -> (#batch, #token, #embedding)
        gru() -> (1, #batch, #hidden)
        '''
        questions = word_embedding(questions)
        _, questions = self.gru(questions)
        questions = questions.transpose(0, 1)
        return questions
```



DMN-片段记忆模块



$Hop = i$

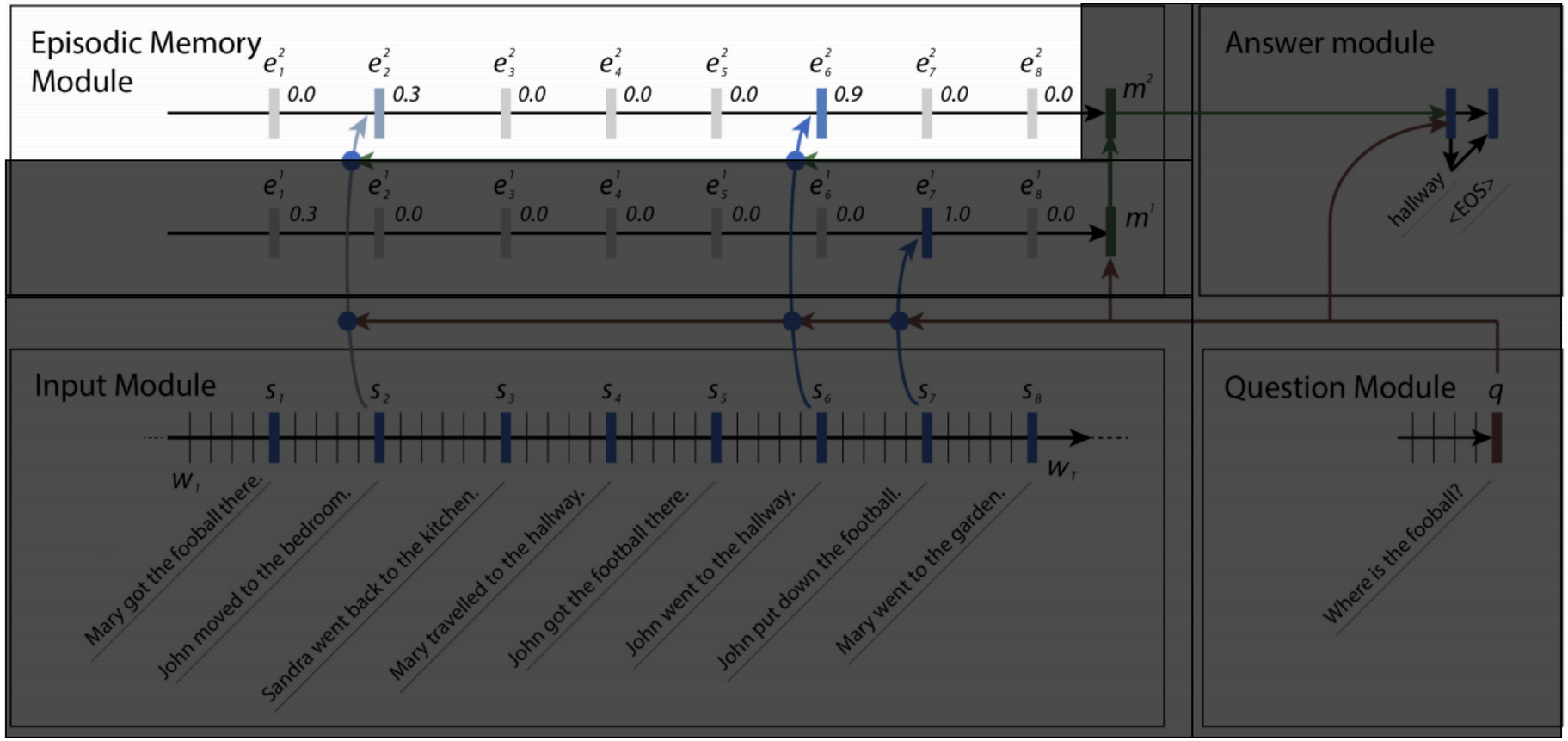
$i = 1$

$$h_t^i = g_t^i \mathrm{GRU}(c_t, h_{t-1}^i) + (1 - g_t^i) h_{t-1}^i$$

$$e^i = h_{T_C}^i$$

Ask Me Anything: Dynamic Memory Networks for Natural Language Processing, Kumar et. al. ICML 2016

DMN-片段记忆模块



$$Hop = i$$

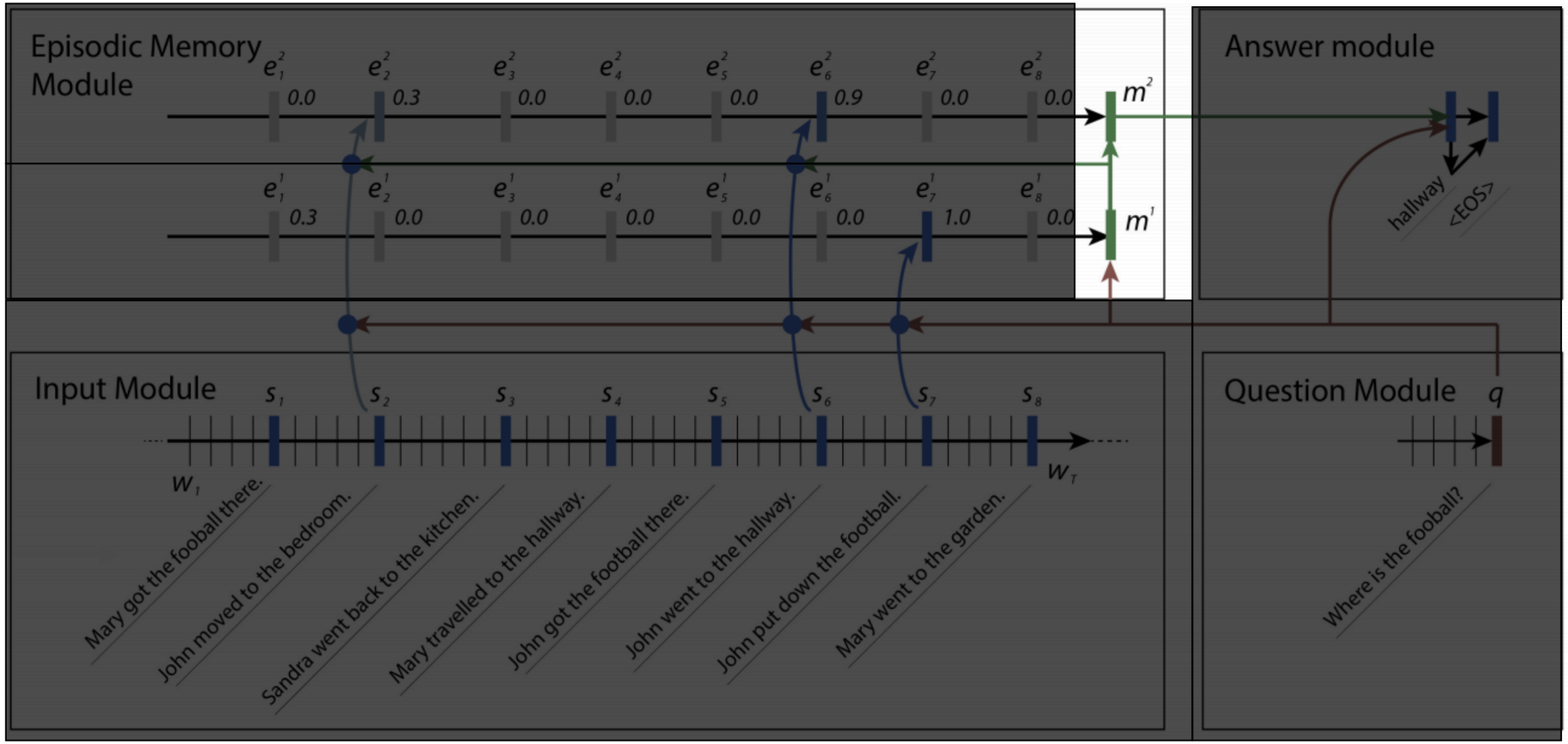
$$i = 2$$

$$h_t^i = g_t^i \mathrm{GRU}(c_t, h_{t-1}^i) + (1 - g_t^i) h_{t-1}^i$$

$$e^i = h_{T_C}^i$$

Ask Me Anything: Dynamic Memory Networks for Natural Language Processing, Kumar et. al. ICML 2016

DMN-片段记忆模块



$$m^i = \text{GRU}(e^i, m^{i-1})$$

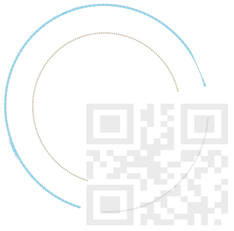
Ask Me Anything: Dynamic Memory Networks for Natural Language Processing, Kumar et. al. ICML 2016

DMN-片段记忆模块

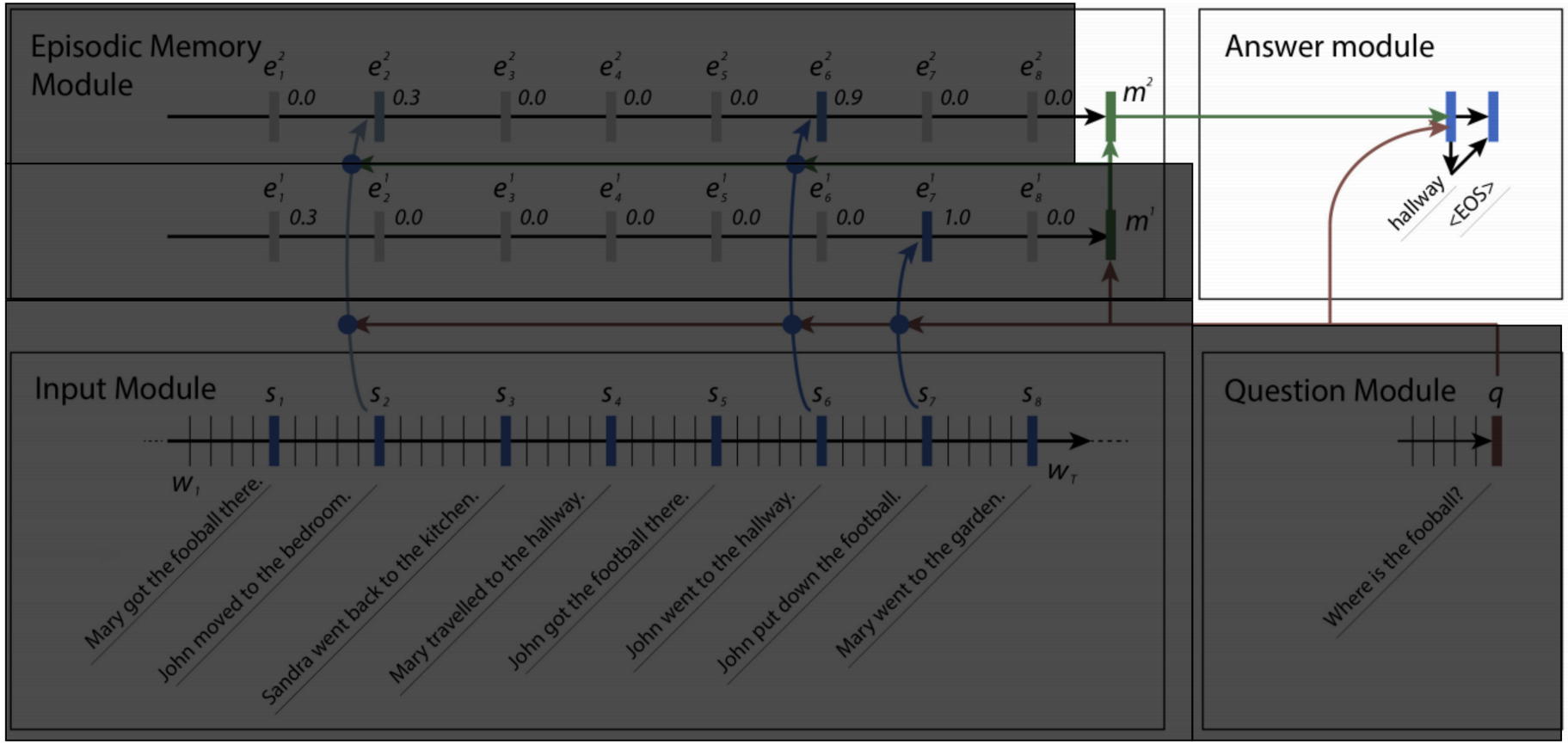
```
class EpisodicMemory(nn.Module):
    def __init__(self, hidden_size):
        super(EpisodicMemory, self).__init__()
        self.AGRU = AttentionGRU(hidden_size, hidden_size)
        self.z1 = nn.Linear(4 * hidden_size, hidden_size)
        self.z2 = nn.Linear(hidden_size, 1)
        self.next_mem = nn.Linear(3 * hidden_size, hidden_size)
        init.xavier_normal(self.z1.state_dict()['weight'])
        init.xavier_normal(self.z2.state_dict()['weight'])
        init.xavier_normal(self.next_mem.state_dict()['weight'])

    def make_interaction(self, facts, questions, prevM):
        '''
        facts.size() -> (#batch, #sentence, #hidden = #embedding)
        questions.size() -> (#batch, 1, #hidden)
        prevM.size() -> (#batch, #sentence = 1, #hidden = #embedding)
        z.size() -> (#batch, #sentence, 4 x #embedding)
        G.size() -> (#batch, #sentence)
        '''
        batch_num, sen_num, embedding_size = facts.size()
        questions = questions.expand_as(facts)
        prevM = prevM.expand_as(facts)

        z = torch.cat([
            facts * questions,
            facts * prevM,
            torch.abs(facts - questions),
            torch.abs(facts - prevM)
        ], dim=2)

        z = z.view(-1, 4 * embedding_size)

        G = F.tanh(self.z1(z))
        G = self.z2(G)
        G = G.view(batch_num, -1)
        G = F.softmax(G)

        return G
```

```
    def forward(self, facts, questions, prevM):
        '''
        facts.size() -> (#batch, #sentence, #hidden = #embedding)
        questions.size() -> (#batch, #sentence = 1, #hidden)
        prevM.size() -> (#batch, #sentence = 1, #hidden = #embedding)
        G.size() -> (#batch, #sentence)
        C.size() -> (#batch, #hidden)
        concat.size() -> (#batch, 3 x #embedding)
        '''
        G = self.make_interaction(facts, questions, prevM)
        C = self.AGRU(facts, G)
        concat = torch.cat([prevM.squeeze(1), C, questions.squeeze(1)], dim=1)
        next_mem = F.relu(self.next_mem(concat))
        next_mem = next_mem.unsqueeze(1)
        return next_mem
```

```
class AttentionGRU(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(AttentionGRU, self).__init__()
        self.hidden_size = hidden_size
        self.AGRUCell = AttentionGRUCell(input_size, hidden_size)

    def forward(self, facts, G):
        '''
        facts.size() -> (#batch, #sentence, #hidden = #embedding)
        fact.size() -> (#batch, #hidden = #embedding)
        G.size() -> (#batch, #sentence)
        g.size() -> (#batch, )
        C.size() -> (#batch, #hidden)
        '''
        batch_num, sen_num, embedding_size = facts.size()
        C = Variable(torch.zeros(self.hidden_size)).cuda()
        for sid in range(sen_num):
            fact = facts[:, sid, :]
            g = G[:, sid]
            if sid == 0:
                C = C.unsqueeze(0).expand_as(fact)
            C = self.AGRUCell(fact, C, g)
        return C
```



DMN-回答模块



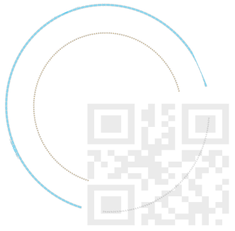
$$y_t = \text{Softmax}(W^{(a)}\alpha_t) \qquad \alpha_0 = m^{T_m}$$

$$\alpha_t = \text{GRU}([y_{t-1}, q], \alpha_{t-1})$$

Ask Me Anything: Dynamic Memory Networks for Natural Language Processing, Kumar et. al. ICML 2016

DMN-回答模块

```
class AnswerModule(nn.Module):
    def __init__(self, vocab_size, hidden_size):
        super(AnswerModule, self).__init__()
        self.z = nn.Linear(2 * hidden_size, vocab_size)
        init.xavier_normal(self.z.state_dict()['weight'])
        self.dropout = nn.Dropout(0.1)

    def forward(self, M, questions):
        M = self.dropout(M)
        concat = torch.cat([M, questions], dim=2).squeeze(1)
        z = self.z(concat)
        return z
```



Dynamic Memory Networks

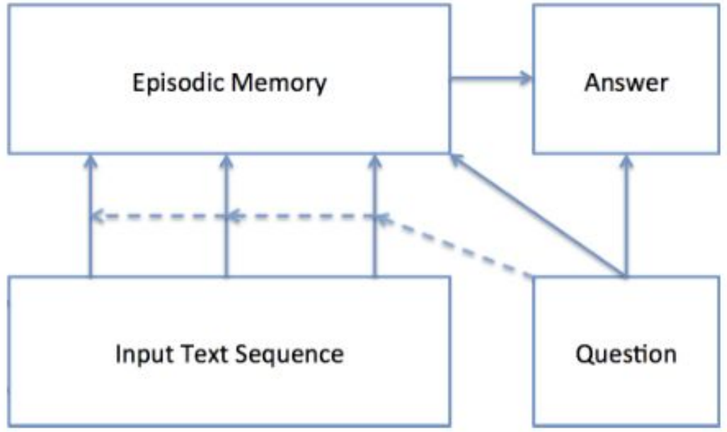


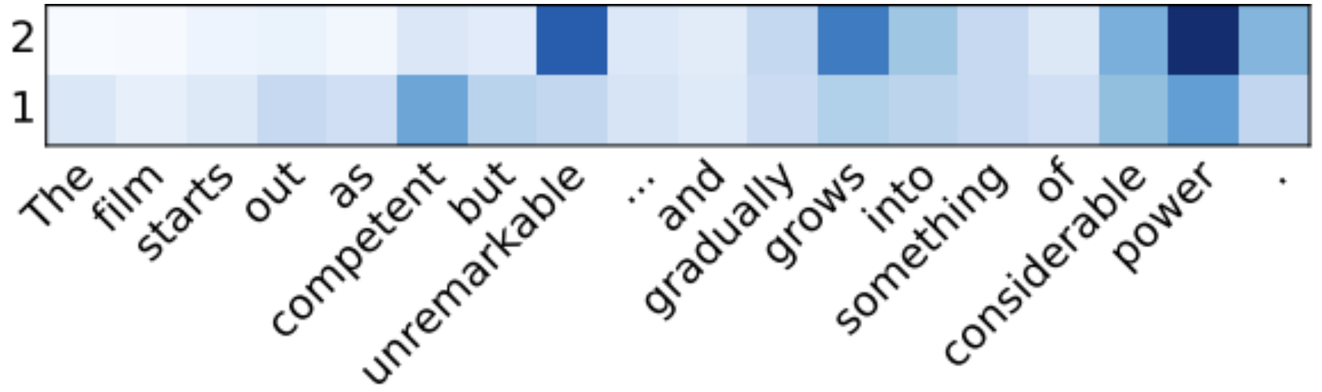
Figure 2. Overview of DMN modules. Communication between them is indicated by arrows and uses vector representations. Questions trigger gates which allow vectors for certain inputs to be given to the episodic memory module. The final state of the episodic memory is the input to the answer module.

```
class DMNPlus(nn.Module):
    def __init__(self, hidden_size, vocab_size, num_hop=3, qa=None):
        super(DMNPlus, self).__init__()
        self.num_hop = num_hop
        self.qa = qa
        self.word_embedding = nn.Embedding(vocab_size, hidden_size, padding_idx=0, sparse=True).cuda()
        init.uniform(self.word_embedding.state_dict()['weight'], a=-(3**0.5), b=3**0.5)
        self.criterion = nn.CrossEntropyLoss(size_average=False)

        self.input_module = InputModule(vocab_size, hidden_size)
        self.question_module = QuestionModule(vocab_size, hidden_size)
        self.memory = EpisodicMemory(hidden_size)
        self.answer_module = AnswerModule(vocab_size, hidden_size)

    def forward(self, contexts, questions):
        '''
        contexts.size() -> (#batch, #sentence, #token) -> (#batch, #sentence, #hidden = #embedding)
        questions.size() -> (#batch, #token) -> (#batch, 1, #hidden)
        '''
        facts = self.input_module(contexts, self.word_embedding)
        questions = self.question_module(questions, self.word_embedding)
        M = questions
        for hop in range(self.num_hop):
            M = self.memory(facts, questions, M)
        preds = self.answer_module(M, questions)
        return preds
```

DMN – Qualitative Results

Question: Where was Mary before the Bedroom?
Answer: Cinema.

Facts	Episode 1	Episode 2	Episode 3
Yesterday Julie traveled to the school.			
Yesterday Marie went to the cinema.		■	
This morning Julie traveled to the kitchen.			
Bill went back to the cinema yesterday.			
Mary went to the bedroom this morning.	■		
Julie went back to the bedroom this afternoon.			
[done reading]			■



Ask Me Anything: Dynamic Memory Networks for Natural Language Processing, Kumar et. al. ICML 2016

VQA

图1. 图中展示了视觉问答的基本形式，图中展示了一位女士鼻子下方胡须的位置挂了两只香蕉，同时图片下方给出了针对这张图片的2个问题（引自：[1]）

AI-Complete Task ?

Image Captioning ?

- A coarse scene-level understanding of an image paired with word n-gram statistics suffices to generate reasonable image captions
- Automatic evaluation is still a difficult and open research problem
- Suggests image captioning may not be as “AI-complete” as desired.
- Source: *https://github.com/karpathy/neuraltalk2*

The man at bat readies to swing at the pitch while the umpire looks on.

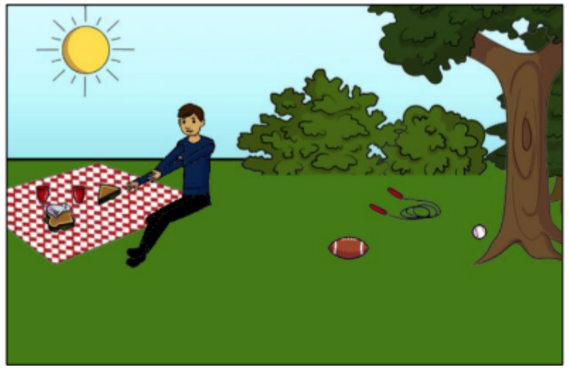
A large bus sitting next to a very tall building.

AI-Complete Task ？

Visual question answering?

Visual questions selectively target different areas of an image , therefore, needs a more detailed understanding of the image

Fine Grained Recognition:
What kind of cheese is on the pizza?



Object Detection:
How many balls are there on the ground?

Commonsense Reasoning :
Does this person have 20/20 vision?

Activity Recognition:
Is the woman sitting?

VQA Dataset: Images

Real Images

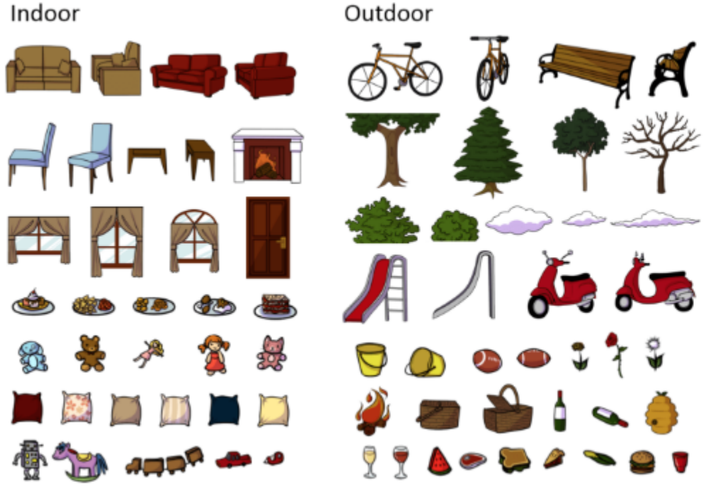
- ❑ Used 204,721 images from the Microsoft Common Objects in Context (MS COCO)
- ❑ Selected images with multiple objects and rich contextual information.

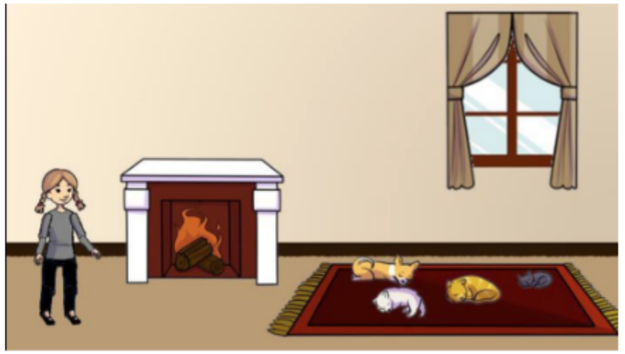



VQA Dataset: Images

Abstract Scenes

- 50,000 scenes
- To enable research focused on high-level reasoning required for VQA, but not the low-level vision tasks.
- Contains 20 “paperdoll” human models with adjustable limbs and over 100 objects and 31 animals in various pose





VQA Demo

How it works

1. You upload an image.
2. Your request is sent to our servers with GPUs courtesy NVIDIA.
3. Our servers run our deep-learning based algorithm.
4. Results and updates are shown in real-time.

Result for Visual Question Answering

Where is the man

Submit

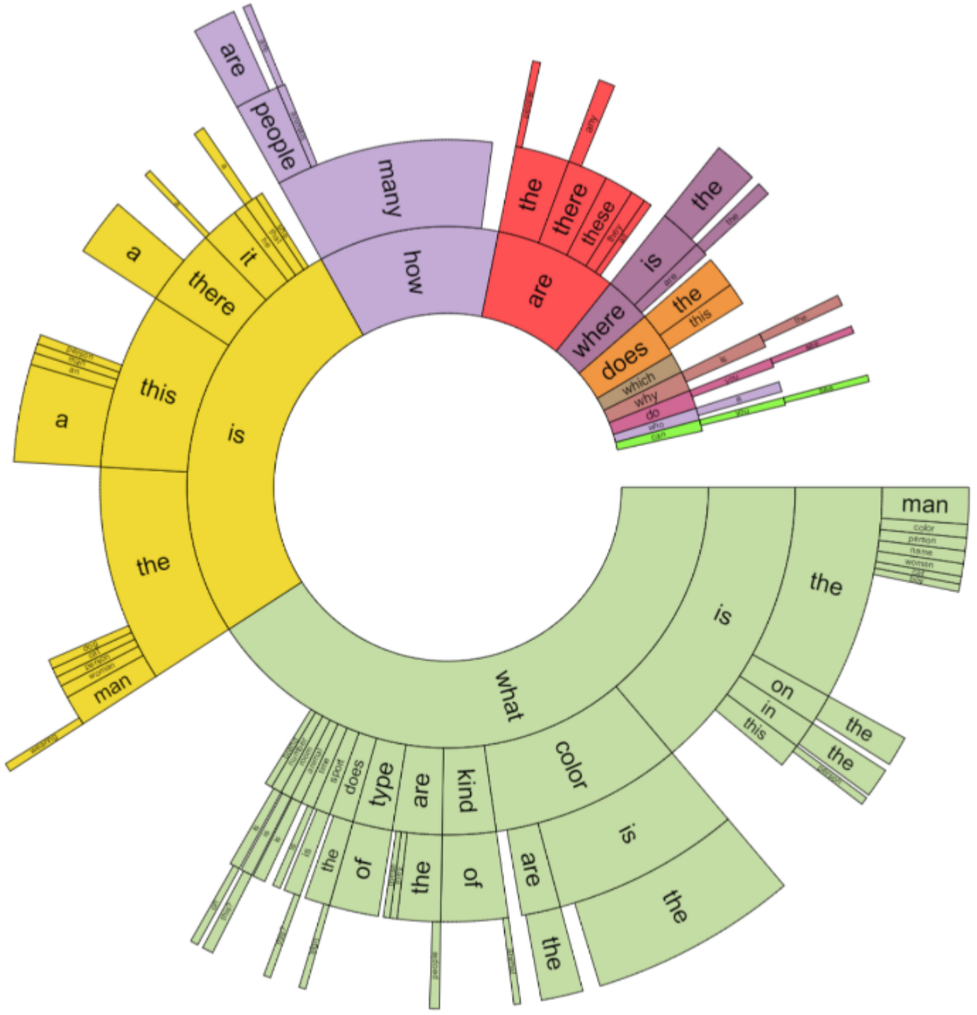
Predicted top-5 answers with confidence:

ocean	99.335%
beach	0.270%
mountain	0.093%
in water	0.063%
water	0.062%

Credits

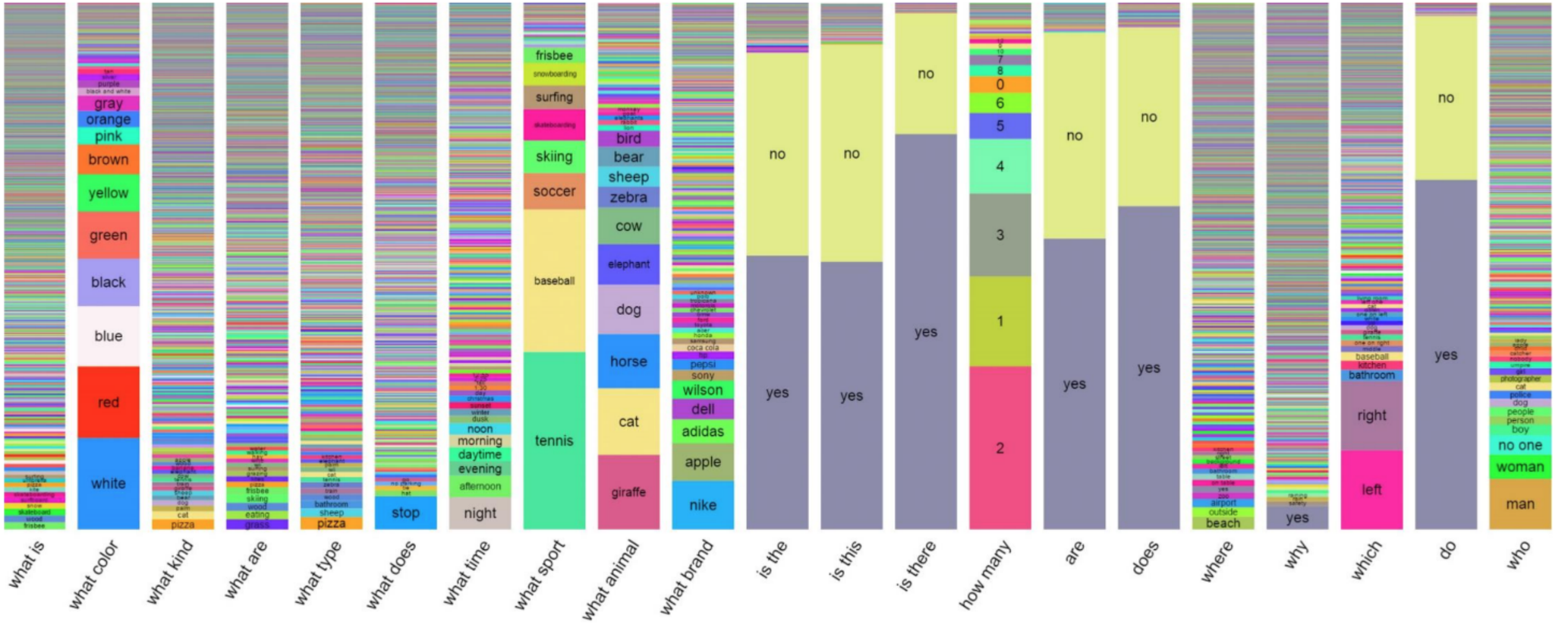
Code for VQA Model
Built by @deshraj

VQA Dataset Analysis: 回答的类型



- Distribution of questions by their ***first four words*** for a random sample of 60K questions
- The ordering of the words starts towards the center and radiates outwards.
- The arc length is proportional to the number of questions containing the word.
- White areas are words with contributions too small to show.

VQA Dataset Analysis: 回答的类型

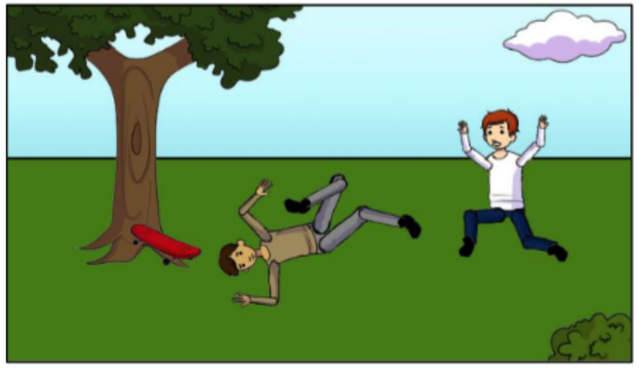


- ❑ “Is the...”, “Are...”, “Does. . . ” : typically answered using “yes” and “no” as answers.
- ❑ “What is. . . ” and “What type. . . ” : have a rich diversity of responses.
- ❑ “What color...” or “Which...” have specialized responses, such as colors, or “left” and “right”.

常识知识: Is the Image Necessary?

- Some questions can be answered correctly using common sense knowledge alone without the need for an image
- What is the color of banana?

How many pickles are on the plate?	1 1 1	1 1 1
What is the shape of the plate?	circle round round	circle round round

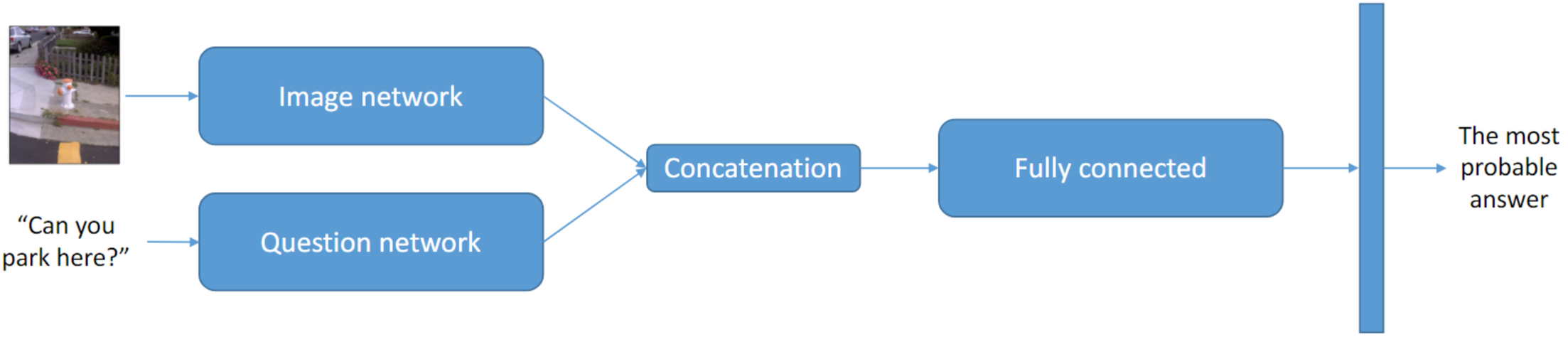


Do you think the boy on the ground has broken legs?	yes yes yes	no no yes
Why is the boy on the right freaking out?	his friend is hurt other boy fell down someone fell	ghost lightning sprayed by hose

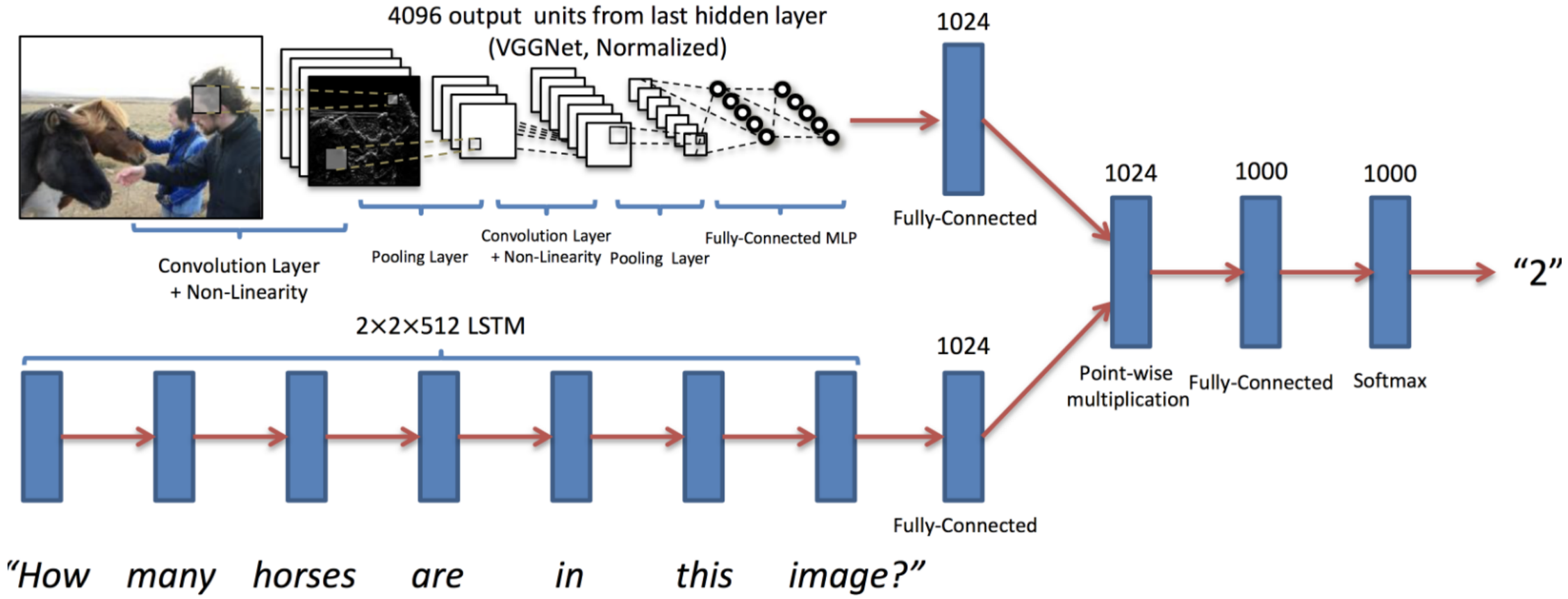
- Black: 问题
- Green: 回答（给定图像）
- Blue: 回答（未给定图像）

Methods

- A 2-channel vision (图像) + language (问题) model
- 选择 top K = 1000 最常见的答案作为可能的输出

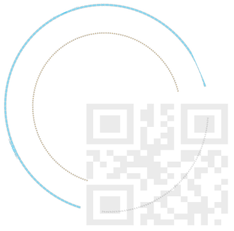


Methods –deeper LSTM Q + norm I



Results

	Open-Ended				Multiple-Choice			
	All	Yes/No	Number	Other	All	Yes/No	Number	Other
prior ("yes")	29.66	70.81	00.39	01.15	29.66	70.81	00.39	01.15
per Q-type prior	37.54	71.03	35.77	09.38	39.45	71.02	35.86	13.34
nearest neighbor	42.70	71.89	24.36	21.94	48.49	71.94	26.00	33.56
BoW Q	48.09	75.66	36.70	27.14	53.68	75.71	37.05	38.64
I	28.13	64.01	00.42	03.77	30.53	69.87	00.45	03.76
BoW Q + I	52.64	75.55	33.67	37.37	58.97	75.59	34.35	50.33
LSTM Q	48.76	78.20	35.68	26.59	54.75	78.22	36.82	38.78
LSTM Q + I	53.74	78.94	35.24	36.42	57.17	78.95	35.80	43.41
deeper LSTM Q	50.39	78.41	34.68	30.03	55.88	78.45	35.91	41.13
deeper LSTM Q + norm I	**57.75**	**80.50**	**36.77**	**43.08**	**62.70**	**80.52**	**38.22**	**53.01**
Caption	26.70	65.50	02.03	03.86	28.29	69.79	02.06	03.82
BoW Q + C	54.70	75.82	40.12	42.56	59.85	75.89	41.16	52.53



Results

What is she playing?
Tennis (0.9972)

What is she holding in her hand?
Tennis racket (0.7395)

What is the color of her shoes?
White (0.48520)

What is the color of her shoe?
Black (0.2661)

How many people are there in the image?
3 (0.2094)

Are they sad?
No (0.8915)

What are the people doing?
Playing frisbee (0.0210)

Where are the people standing?
Frisbee (0.0541)

Where are they?
Park (0.0669)

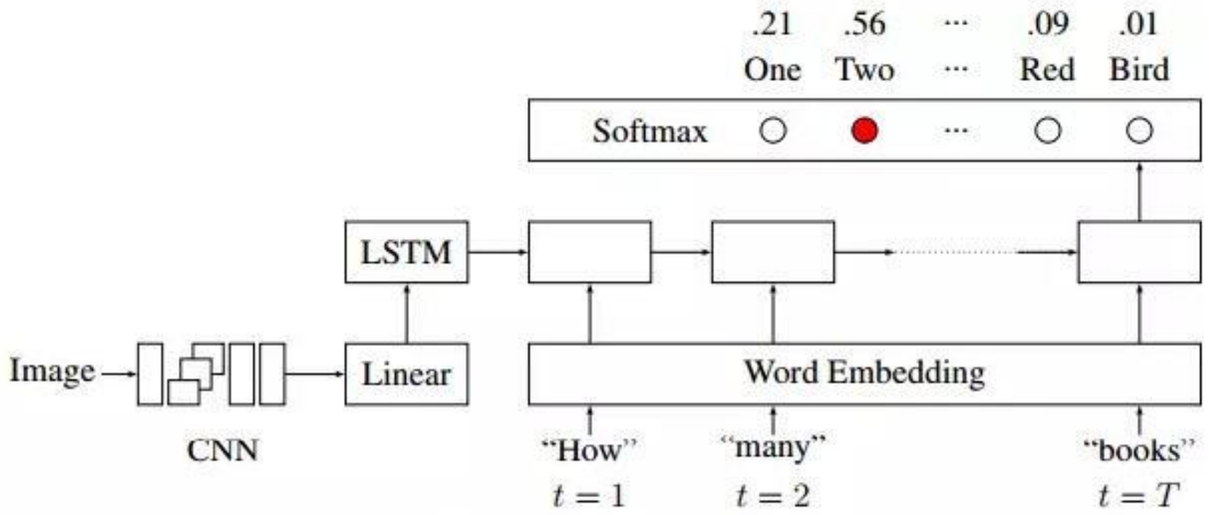
Are they happy?
Yes (0.9536)

Why are they upset?
Happy (0.1090)

What are they doing?
Skateboarding (0.0480)

VIS+LSTM

文中模型的基本结构是首先使用CNN抽取图片信息，完成之后接LSTM产生预测结果。
Mengye Ren等人考虑到由于没有一个完好的评价答案句子精确度的标准，因此他们将注意力集中在有限的领域问题，这些问题可以用一个单词作为视觉问答的答案，这样就可以把视觉问答视为一个多分类问题，从而可以利用现有的精确度评价标准度量答案。



VIS+LSTM

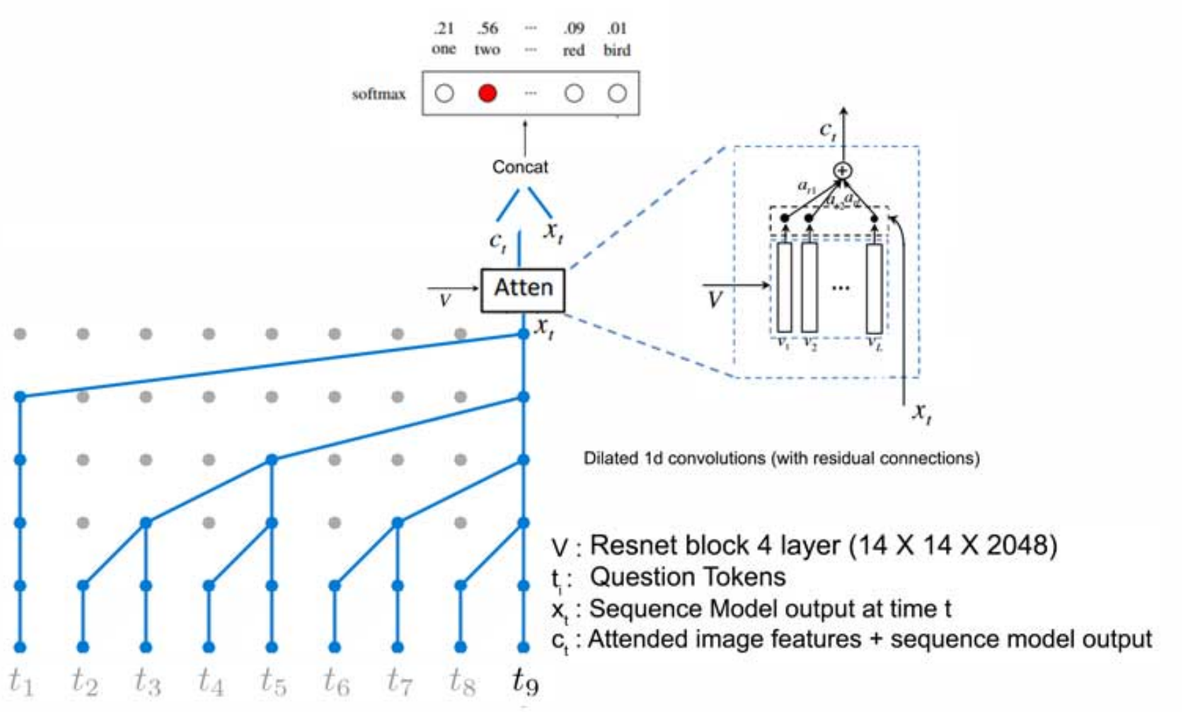
文中模型的基本结构是首先使用CNN抽取图片信息，完成之后接LSTM产生预测结果。

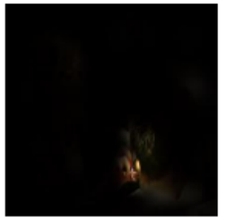
Table 2: DAQUAR and COCO-QA results

	DAQUAR			COCO-QA		
	Acc.	WUPS 0.9	WUPS 0.0	Acc.	WUPS 0.9	WUPS 0.0
MULTI-WORLD [32]	0.1273	0.1810	0.5147	-	-	-
GUESS	0.1824	0.2965	0.7759	0.0730	0.1837	0.7413
BOW	0.3267	0.4319	0.8130	0.3752	0.4854	0.8278
LSTM	0.3273	0.4350	0.8162	0.3676	0.4758	0.8234
IMG	-	-	-	0.4302	0.5864	0.8585
IMG+PRIOR	-	-	-	0.4466	0.6020	0.8624
K-NN (K=31, 13)	0.3185	0.4242	0.8063	0.4496	0.5698	0.8557
IMG+BOW	0.3417	0.4499	0.8148	**0.5592**	**0.6678**	**0.8899**
VIS+LSTM	0.3441	0.4605	**0.8223**	0.5331	0.6391	0.8825
ASK-NEURON [14]	0.3468	0.4076	0.7954	-	-	-
2-VIS+BLSTM	**0.3578**	**0.4683**	0.8215	0.5509	0.6534	0.8864
FULL	**0.3694**	**0.4815**	**0.8268**	**0.5784**	**0.6790**	**0.8952**
HUMAN	0.6027	0.6104	0.7896	-	-	-

Fully Convolutional Visual Question Answering

This is an attention based model for VQA using a dilated convolutional neural network for modelling the question and a resnet for visual features.
The text model is based on the recent convolutional architecture ByteNet.
Stacked attention distributions over the images are then used to compute weighted image features, which are concatenated with the text features to predict the answer.



	is she going to eat both pizza			No
	What color is the traffic light			green
	is the persons hair short			Yes

Visual Dialogue

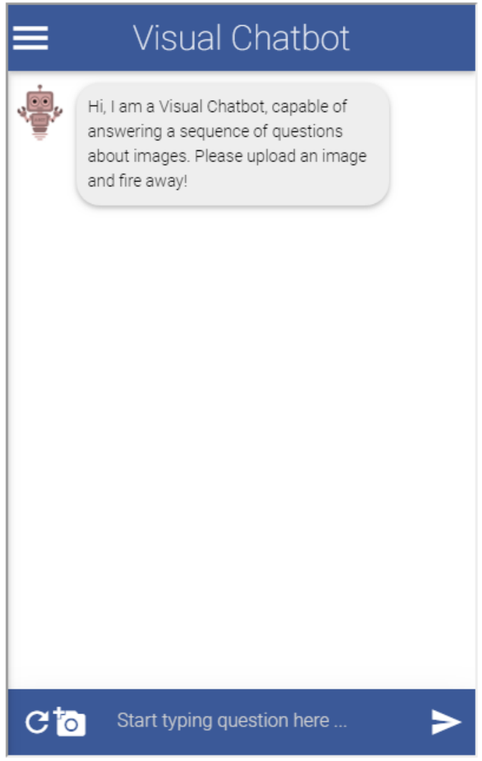
Visual Dialog Overview People Data Bibtex Acknowledgements VirginiaTech Georgia Tech

Specifically, given an image, a dialog history, and a follow-up question about the image, the agent has to answer the question.

- 10 rounds of question-answers / dialog
- Total 1.2M dialog question-answers

Apr 2017 — Torch code for training/evaluating Visual Dialog models, pretrained models and Visual Chatbot demo are now available!
Mar 2017 — VisDial v0.9 dataset and code for real-time chat interface used to collect data on AMT are now available!

Visual Chatbot demo



Email — contact@visualdialog.org

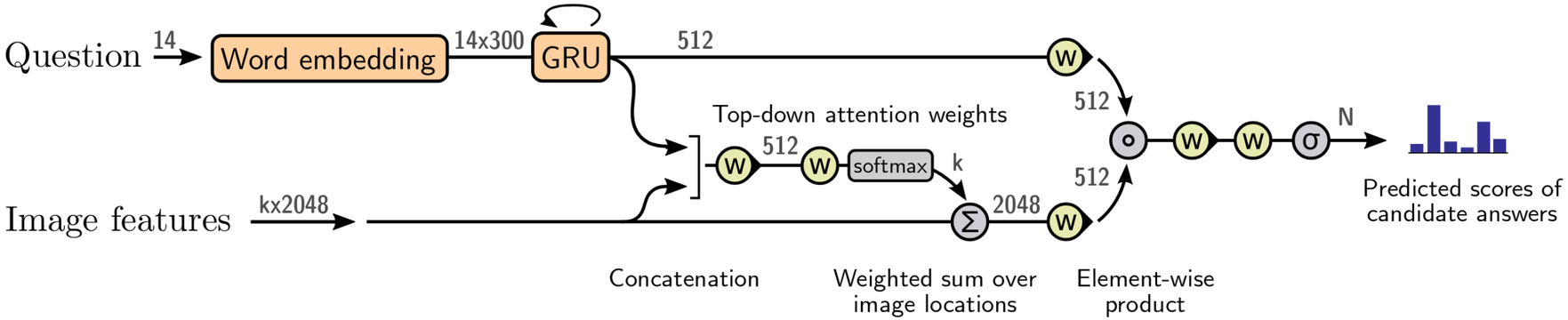
Subscribe for Visual Dialog release updates

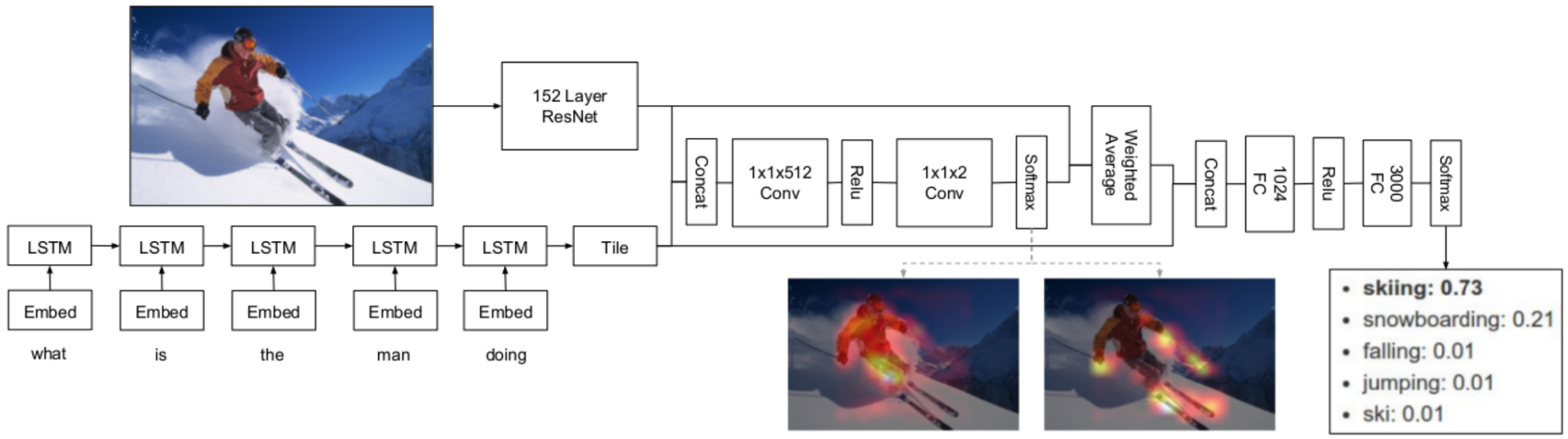
https://visualdialog.org/

Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering

This paper solves two tasks: Image Captioning and VQA. The main idea is to use Faster R-CNN to embed images (kx2048 from k bounding boxes) instead of ResNet (14x14x2048) and apply attention over k vectors.

For VQA, this is basically (Faster R-CNN + Show Attend Ask Answer). SAAA(Show Ask Attend Answer) calculates a 2D attention map from the concatenation of a text vector (2048-dim from LSTM) and image tensor (2048x14x14 from ResNet).
This image feature can be thought as a collection of 2048-dim feature vectors. This paper uses Faster R-CNN to get k bounding boxes. Each bounding box is a 2048-dim vector so we have kx2048, which is fed to SAAA.

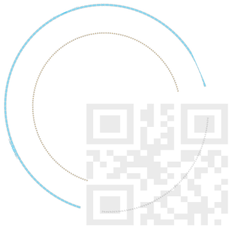




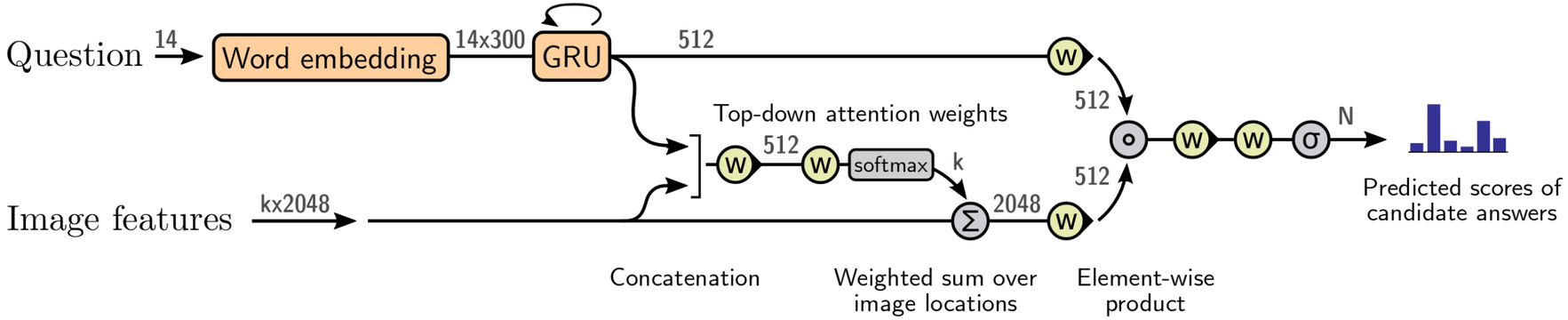
Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering

```
## Script for downloading data

# GloVe Vectors
wget -P data http://nlp.stanford.edu/data/glove.6B.zip
unzip data/glove.6B.zip -d data/glove
rm data/glove.6B.zip

# Questions
wget -P data http://visualqa.org/data/mscoco/vqa/v2_Questions_Train_mscoco.zip
unzip data/v2_Questions_Train_mscoco.zip -d data
rm data/v2_Questions_Train_mscoco.zip

wget -P data http://visualqa.org/data/mscoco/vqa/v2_Questions_Val_mscoco.zip
unzip data/v2_Questions_Val_mscoco.zip -d data
rm data/v2_Questions_Val_mscoco.zip

wget -P data http://visualqa.org/data/mscoco/vqa/v2_Questions_Test_mscoco.zip
unzip data/v2_Questions_Test_mscoco.zip -d data
rm data/v2_Questions_Test_mscoco.zip

# Annotations
wget -P data http://visualqa.org/data/mscoco/vqa/v2_Annotations_Train_mscoco.zip
unzip data/v2_Annotations_Train_mscoco.zip -d data
rm data/v2_Annotations_Train_mscoco.zip

wget -P data http://visualqa.org/data/mscoco/vqa/v2_Annotations_Val_mscoco.zip
unzip data/v2_Annotations_Val_mscoco.zip -d data
rm data/v2_Annotations_Val_mscoco.zip

# Image Features
wget -P data https://storage.googleapis.com/bottom-up-attention/trainval_36.zip
unzip data/trainval_36.zip -d data
rm data/trainval_36.zip
```

```
      4096 Apr 14 06:04 ./
      4096 Apr 16 06:24 ../
      4096 Apr 14 05:56 cache/
    605752 Apr 14 05:54 dictionary.pkl
      4096 Apr 13 17:55 glove/
  23881328 Apr 14 05:54 glove6b_init_300d.npy
24532909760 Apr 14 06:04 train36.hdf5
   1297643 Apr 14 06:04 train36_imgid2idx.pkl
    729298 Apr  6 13:34 train_ids.pkl
      4096 Aug 14  2017 trainval_36/
 355516514 Apr 26  2017 v2_mscoco_train2014_annotations.json
 171762489 Apr 26  2017 v2_mscoco_val2014_annotations.json
  41800665 Apr 26  2017 v2_OpenEnded_mscoco_test2015_questions.json
  10032875 Apr 26  2017 v2_OpenEnded_mscoco_test-dev2015_questions.json
  41957627 Apr 26  2017 v2_OpenEnded_mscoco_train2014_questions.json
  20247670 Apr 26  2017 v2_OpenEnded_mscoco_val2014_questions.json
12003443552 Apr 14 06:04 val36.hdf5
    629212 Apr 14 06:04 val36_imgid2idx.pkl
    356820 Apr  6 13:34 val_ids.pkl
```

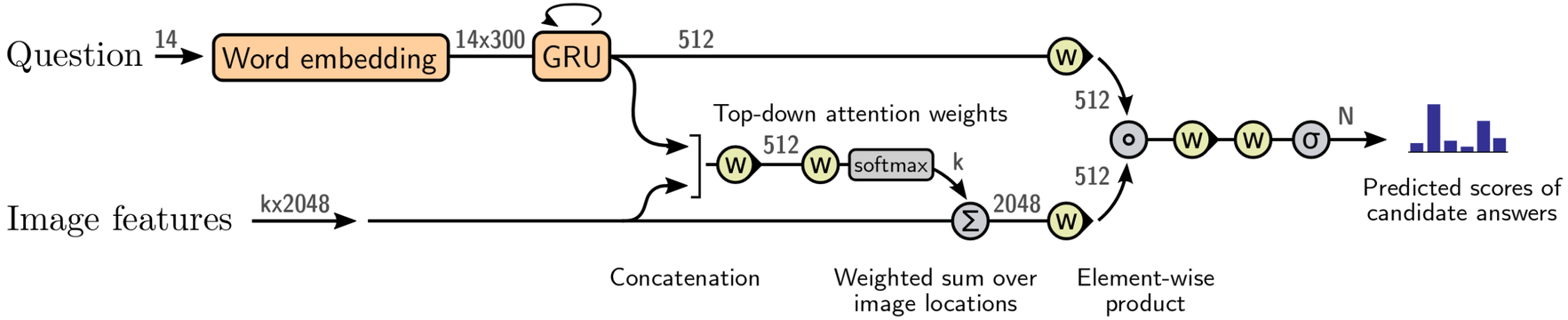


Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering



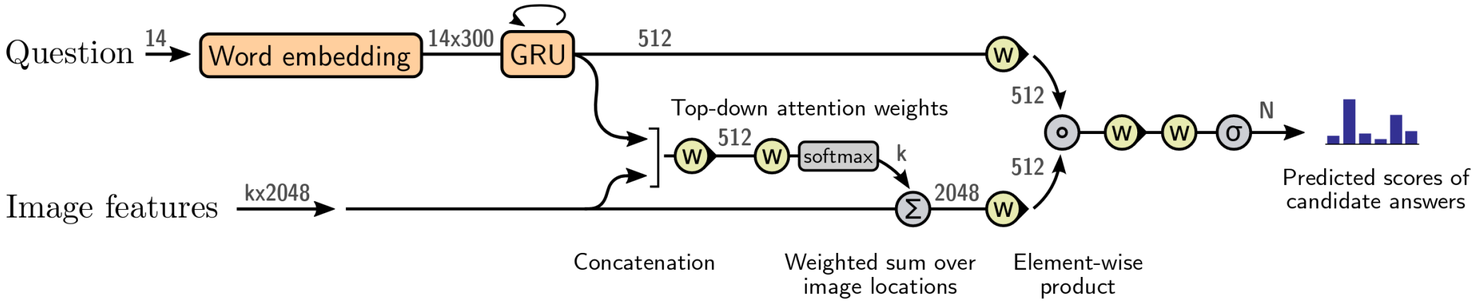

```
def build_baseline0(dataset, num_hid):
    w_emb = WordEmbedding(dataset.dictionary.ntoken, 300, 0.0)
    q_emb = QuestionEmbedding(300, num_hid, 1, False, 0.0)
    v_att = Attention(dataset.v_dim, q_emb.num_hid, num_hid)
    q_net = FCNet([num_hid, num_hid])
    v_net = FCNet([dataset.v_dim, num_hid])
    classifier = SimpleClassifier(
        num_hid, 2 * num_hid, dataset.num_ans_candidates, 0.5)
    return BaseModel(w_emb, q_emb, v_att, q_net, v_net, classifier)
```

Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering



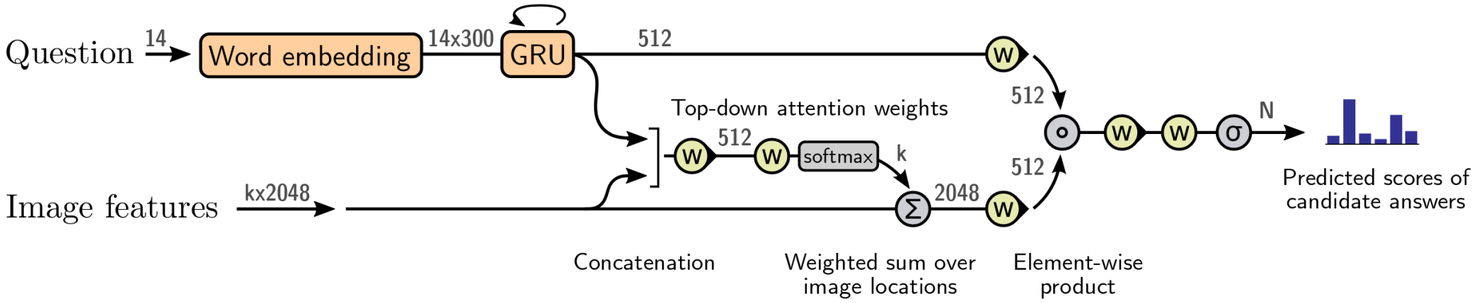

```python
class Attention(nn.Module):
    def __init__(self, v_dim, q_dim, num_hid):
        super(Attention, self).__init__()
        self.nonlinear = FCNet([v_dim + q_dim, num_hid])
        self.linear = weight_norm(nn.Linear(num_hid, 1), dim=None)

    def forward(self, v, q):
        """
        v: [batch, k, vdim]
        q: [batch, qdim]
        """
        logits = self.logits(v, q)
        w = nn.functional.softmax(logits, 1)
        return w

    def logits(self, v, q):
        num_objs = v.size(1)
        q = q.unsqueeze(1).repeat(1, num_objs, 1)
        vq = torch.cat((v, q), 2)
        joint_repr = self.nonlinear(vq)
        logits = self.linear(joint_repr)
        return logits
```

Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering



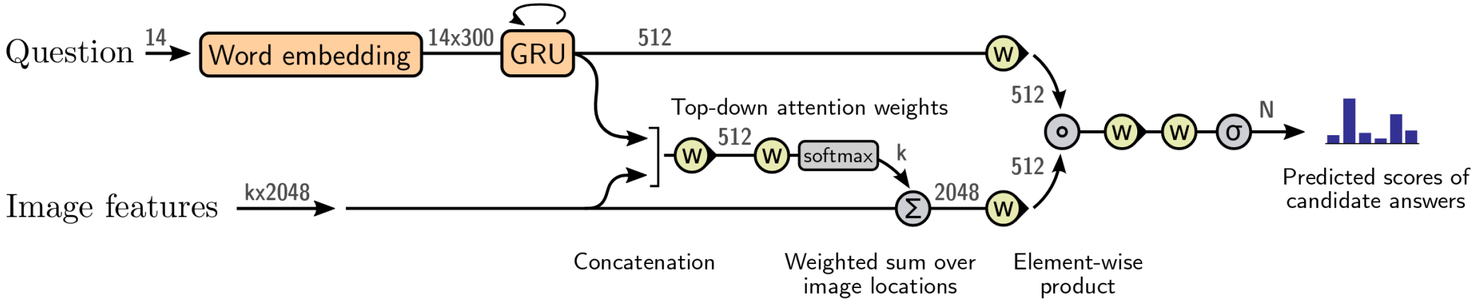

```python
def train(model, train_loader, eval_loader, num_epochs, output):
    utils.create_dir(output)
    optim = torch.optim.Adamax(model.parameters())
    logger = utils.Logger(os.path.join(output, 'log.txt'))
    best_eval_score = 0

    for epoch in range(num_epochs):
        total_loss = 0
        train_score = 0
        t = time.time()

        for i, (v, b, q, a) in enumerate(train_loader):
            v = Variable(v).cuda()
            b = Variable(b).cuda()
            q = Variable(q).cuda()
            a = Variable(a).cuda()

            pred = model(v, b, q, a)
            loss = instance_bce_with_logits(pred, a)
            loss.backward()
            nn.utils.clip_grad_norm(model.parameters(), 0.25)
            optim.step()
            optim.zero_grad()

            batch_score = compute_score_with_logits(pred, a.data).sum()
            total_loss += loss.data[0] * v.size(0)
            train_score += batch_score

        total_loss /= len(train_loader.dataset)
        train_score = 100 * train_score / len(train_loader.dataset)
        model.train(False)
        eval_score, bound = evaluate(model, eval_loader)
        model.train(True)

        logger.write('epoch %d, time: %.2f' % (epoch, time.time()-t))
        logger.write('\ttrain_loss: %.2f, score: %.2f' % (total_loss, train_score))
        logger.write('\teval score: %.2f (%.2f)' % (100 * eval_score, 100 * bound))

        if eval_score > best_eval_score:
            model_path = os.path.join(output, 'model.pth')
            torch.save(model.state_dict(), model_path)
            best_eval_score = eval_score
```

Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering



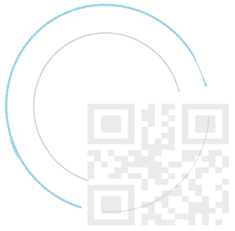

```python
def evaluate(model, dataloader):
    score = 0
    upper_bound = 0
    num_data = 0
    for v, b, q, a in iter(dataloader):
        v = Variable(v, volatile=True).cuda()
        b = Variable(b, volatile=True).cuda()
        q = Variable(q, volatile=True).cuda()
        pred = model(v, b, q, None)
        batch_score = compute_score_with_logits(pred, a.cuda()).sum()
        score += batch_score
        upper_bound += (a.max(1)[0]).sum()
        num_data += pred.size(0)

    score = score / len(dataloader.dataset)
    upper_bound = upper_bound / len(dataloader.dataset)
    return score, upper_bound
```

Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering

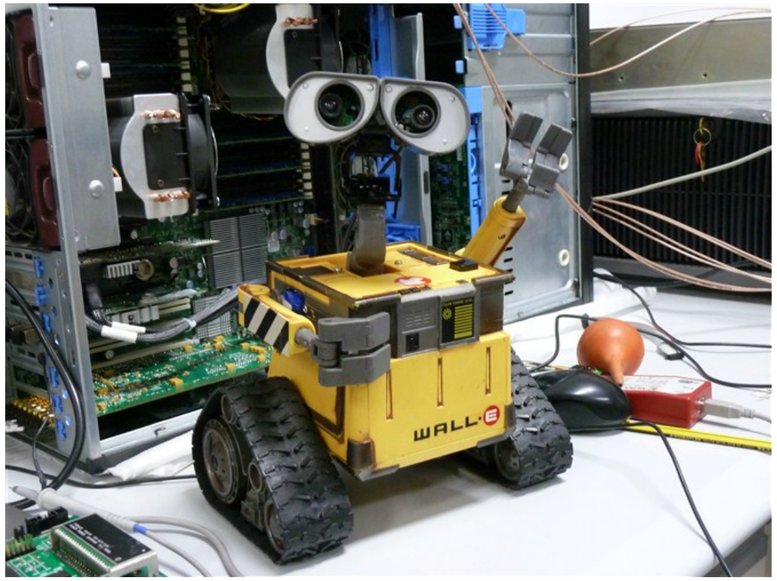



```python
if __name__ == '__main__':
    args = parse_args()

    torch.manual_seed(args.seed)
    torch.cuda.manual_seed(args.seed)
    torch.backends.cudnn.benchmark = True

    dictionary = Dictionary.load_from_file('data/dictionary.pkl')
    train_dset = VQAFeatureDataset('train', dictionary)
    eval_dset = VQAFeatureDataset('val', dictionary)
    batch_size = args.batch_size

    constructor = 'build_%s' % args.model
    model = getattr(base_model, constructor)(train_dset, args.num_hid).cuda()
    model.w_emb.init_embedding('data/glove6b_init_300d.npy')

    model = nn.DataParallel(model).cuda()

    train_loader = DataLoader(train_dset, batch_size, shuffle=True, num_workers=1)
    eval_loader =  DataLoader(eval_dset, batch_size, shuffle=True, num_workers=1)
    train(model, train_loader, eval_loader, args.epochs, args.output)
```

```
epoch 0, time: 1923.77
        train_loss: 9.67, score: 40.09
        eval score: 50.93 (92.66)
epoch 1, time: 248.82
        train_loss: 3.91, score: 52.98
        eval score: 56.06 (92.66)
epoch 2, time: 250.93
        train_loss: 3.57, score: 57.46
        eval score: 58.76 (92.66)
epoch 3, time: 255.43
        train_loss: 3.37, score: 60.51
        eval score: 60.12 (92.66)
epoch 4, time: 253.58
        train_loss: 3.22, score: 62.70
        eval score: 61.15 (92.66)
epoch 5, time: 252.91
        train_loss: 3.09, score: 64.71
        eval score: 61.86 (92.66)
epoch 6, time: 252.50
        train_loss: 2.99, score: 66.44
        eval score: 62.36 (92.66)
epoch 7, time: 252.14
        train_loss: 2.89, score: 67.89
        eval score: 62.59 (92.66)
epoch 8, time: 248.87
        train_loss: 2.81, score: 69.39
        eval score: 62.98 (92.66)
epoch 9, time: 249.60
        train_loss: 2.73, score: 70.75
        eval score: 63.08 (92.66)
epoch 10, time: 248.71
        train_loss: 2.65, score: 72.09
        eval score: 63.17 (92.66)
epoch 11, time: 246.01
        train_loss: 2.59, score: 73.25
        eval score: 63.42 (92.66)
epoch 12, time: 255.00
        train_loss: 2.52, score: 74.31
        eval score: 63.44 (92.66)
epoch 13, time: 258.26
        train_loss: 2.46, score: 75.36
        eval score: 63.45 (92.66)
epoch 14, time: 253.48
        train_loss: 2.41, score: 76.32
        eval score: 63.49 (92.66)
epoch 15, time: 259.66
        train_loss: 2.35, score: 77.14
        eval score: 63.45 (92.66)
epoch 16, time: 255.82
        train_loss: 2.31, score: 77.97
        eval score: 63.47 (92.66)
epoch 17, time: 257.74
        train_loss: 2.26, score: 78.56
        eval score: 63.45 (92.66)
epoch 18, time: 258.76
        train_loss: 2.22, score: 79.28
        eval score: 63.43 (92.66)
epoch 19, time: 257.84
        train_loss: 2.18, score: 79.81
```




EXID



身体和灵魂，总有一个在路上

Wish you all the best in your future endeavors