Model Overview and Memory Networks

By Keat Loong Chia, Ming Liang , Stephanie Lew and Luong Quoc Trung

Tree LSTMs and QRNNs

Tree LSTMs

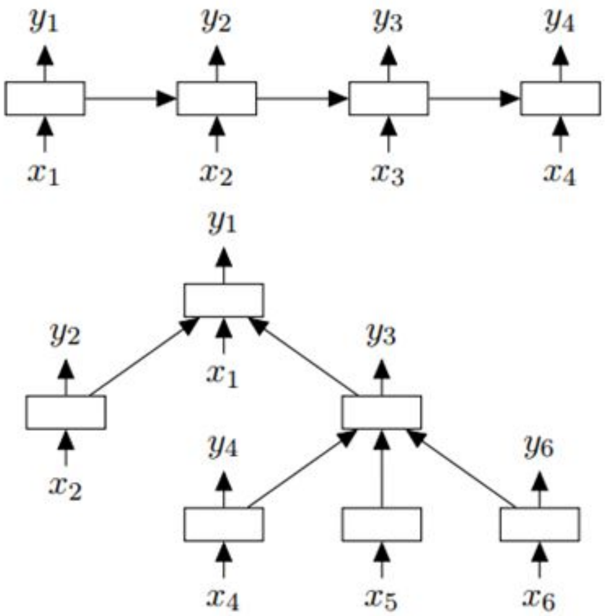


Figure 1: **Top:** A chain-structured LSTM network. **Bottom:** A tree-structured LSTM network with arbitrary branching factor.

- Compared to a normal LSTM (essentially a linear chain), where each unit only takes in the hidden state from the time period before, the Tree LSTM is able to take hidden states from arbitrarily many child units
- The Tree LSTM can be thought of as a more generalized version of the standard LSTM structure. The standard LSTM will just be a special case of the Tree LSTM where each internal node has only one child
- Difference: Taking in the hidden states from more than one node

Tree LSTMs

Child-Sum Tree-LSTM

$$\boxed{\tilde{h}_j = \sum_{k \in C(j)} h_k, \quad (2)}$$

Hidden state is sum of all child nodes' hidden states

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right), \quad (3)$$

$$\boxed{f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right), \quad (4)}$$

Uses the same weight matrix for all child nodes

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right), \quad (5)$$

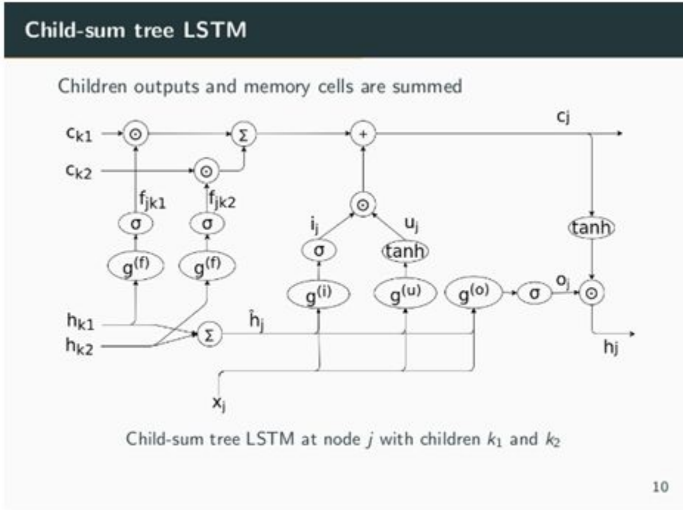
$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right), \quad (6)$$

$$\boxed{c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \quad (7)}$$

Memory from child nodes is the sum of all child hidden states multiplied by forget gate values

$$h_j = o_j \odot \tanh(c_j), \quad (8)$$

- Does not account for children order
- Works with variable number of children
- Shares gate weights between children
- E.g. Dependency Tree-LSTM



Tree LSTMs

N-ary Tree LSTM

$$i_j = \sigma\left(W^{(i)} x_j + \sum_{\ell=1}^{N} U_\ell^{(i)} h_{j\ell} + b^{(i)}\right), \quad (9)$$

$$\boxed{f_{jk} = \sigma\left(W^{(f)} x_j + \sum_{\ell=1}^{N} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right),} \quad (10)$$

$$o_j = \sigma\left(W^{(o)} x_j + \sum_{\ell=1}^{N} U_\ell^{(o)} h_{j\ell} + b^{(o)}\right), \quad (11)$$

$$u_j = \tanh\left(W^{(u)} x_j + \sum_{\ell=1}^{N} U_\ell^{(u)} h_{j\ell} + b^{(u)}\right), \quad (12)$$

$$c_j = i_j \odot u_j + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell}, \quad (13)$$

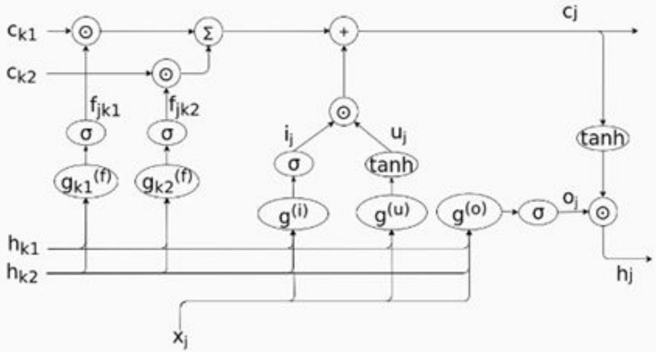
$$h_j = o_j \odot \tanh(c_j), \quad (14)$$

Uses a different weight matrix for each child node

- Each node must have only at most N children
- Finer control over how much information propagates

N-ary tree LSTM

Given $g_k^{(n)}(x_t, h_{l_1}, \cdots, h_{l_N}) = W^{(n)} x_t + \sum_{l=1}^{N} U_{kl}^{(n)} h_{jl} + b^{(n)}$



Binary tree LSTM at node j with children k_1 and k_2

Q-RNNs

RNNs: Slow and cannot be run in parallel, as each time step depends on the previous time step's computation

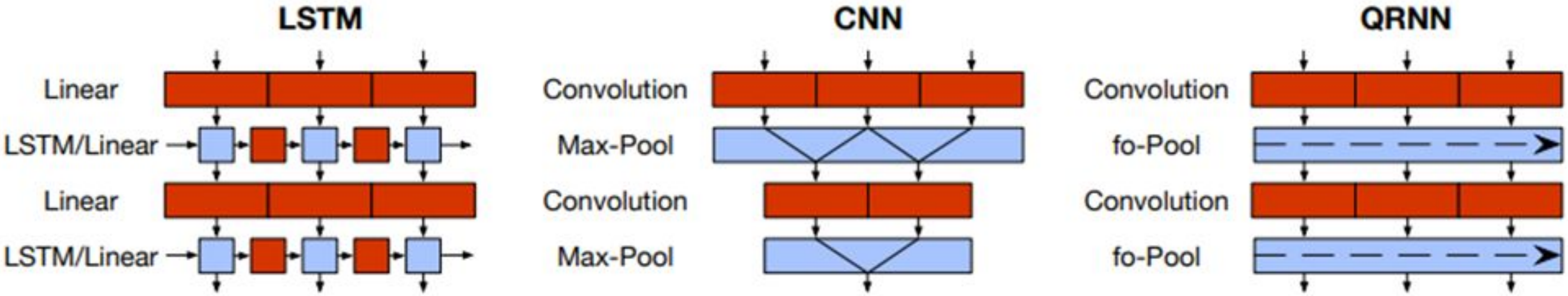
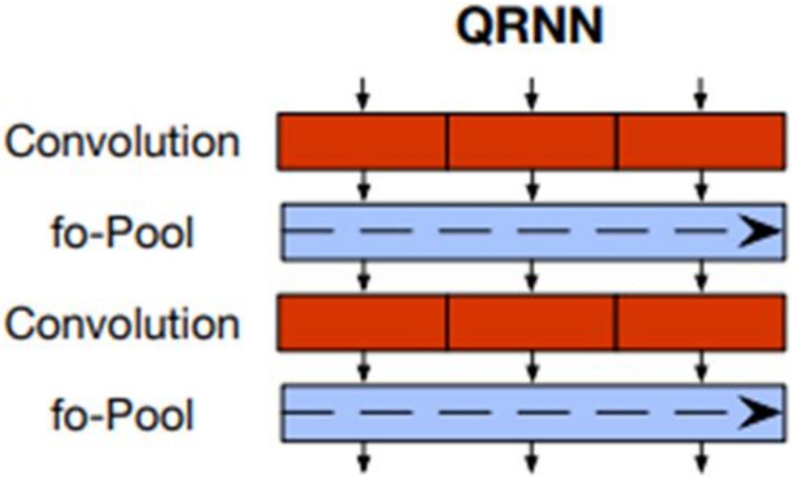


Figure 1: Block diagrams showing the computation structure of the QRNN compared with typical LSTM and CNN architectures. Red signifies convolutions or matrix multiplications; a continuous block means that those computations can proceed in parallel. Blue signifies parameterless functions that operate in parallel along the channel/feature dimension. LSTMs can be factored into (red) linear blocks and (blue) elementwise blocks, but computation at each timestep still depends on the results from the previous timestep.

Q-RNNs

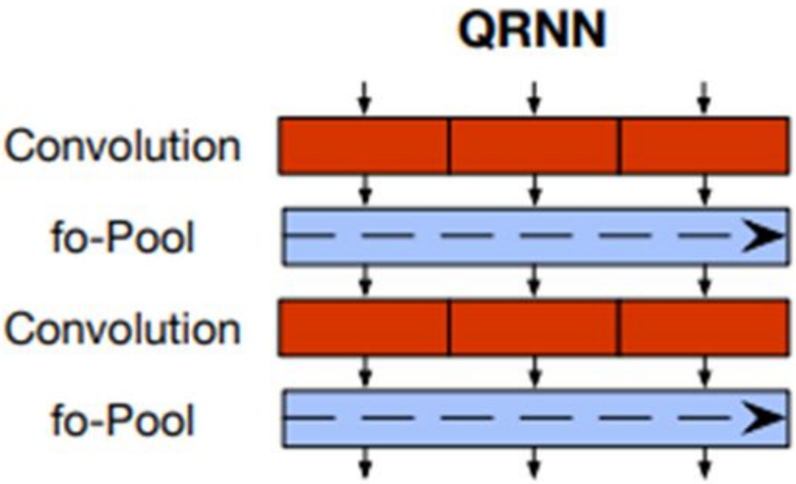


Convolutional layer (filter width 2)

$$\mathbf{z}_t = \tanh(\mathbf{W}_z^1 \mathbf{x}_{t-1} + \mathbf{W}_z^2 \mathbf{x}_t)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f^1 \mathbf{x}_{t-1} + \mathbf{W}_f^2 \mathbf{x}_t)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o^1 \mathbf{x}_{t-1} + \mathbf{W}_o^2 \mathbf{x}_t).$$

- Only takes in inputs at or before time step t
- Each z_t, f_t or o_t only depends on input vectors from time steps before it
- Does not require any outputs from previous time step
- Easily parallelizable

Q-RNNs



Pooling Layer

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t,$$ f-Pooling

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t.$$ fo-Pooling

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{z}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t.$$ ifo-Pooling

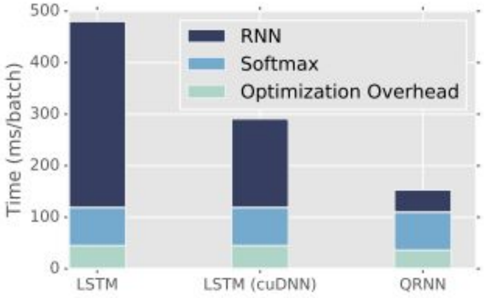
- 3 different pooling options
- Must be calculated for each time step in sequence
- But simple to calculate and can be parallelized over feature dimensions

Q-RNNs: Language Modeling

- Better

Model	Parameters	Validation	Test
LSTM (medium) (Zaremba et al., 2014)	20M	86.2	82.7
Variational LSTM (medium) (Gal & Ghahramani, 2016)	20M	81.9	79.7
LSTM with CharCNN embeddings (Kim et al., 2016)	19M	–	78.9
Zoneout + Variational LSTM (medium) (Merity et al., 2016)	20M	84.4	80.6
Our models			
LSTM (medium)	20M	85.7	82.0
QRNN (medium)	18M	82.9	79.9
QRNN + zoneout ($p = 0.1$) (medium)	18M	82.1	78.3

- Faster



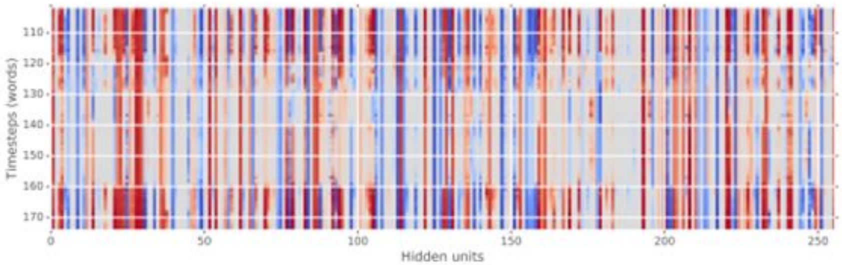
		Sequence length				
		32	64	128	256	512
Batch size	8	**5.5x**	**8.8x**	**11.0x**	**12.4x**	**16.9x**
	16	**5.5x**	**6.7x**	**7.8x**	**8.3x**	**10.8x**
	32	**4.2x**	**4.5x**	**4.9x**	**4.9x**	**6.4x**
	64	**3.0x**	**3.0x**	**3.0x**	**3.0x**	**3.7x**
	128	**2.1x**	**1.9x**	**2.0x**	**2.0x**	**2.4x**
	256	**1.4x**	**1.4x**	**1.3x**	**1.3x**	**1.3x**

Q-RNNs: Sentiment Analysis

- Often better and faster than LSTMs

Model	Time / Epoch (s)	Test Acc (%)
BSVM-bi (Wang & Manning, 2012)	–	91.2
2 layer sequential BoW CNN (Johnson & Zhang, 2014)	–	92.3
Ensemble of RNNs and NB-SVM (Mesnil et al., 2014)	–	92.6
2-layer LSTM (Longpre et al., 2016)	–	87.6
Residual 2-layer bi-LSTM (Longpre et al., 2016)	–	90.1
Our models		
Deeply connected 4-layer LSTM (cuDNN optimized)	480	90.9
Deeply connected 4-layer QRNN	150	91.4
D.C. 4-layer QRNN with $k = 4$	160	91.1

- More interpretable



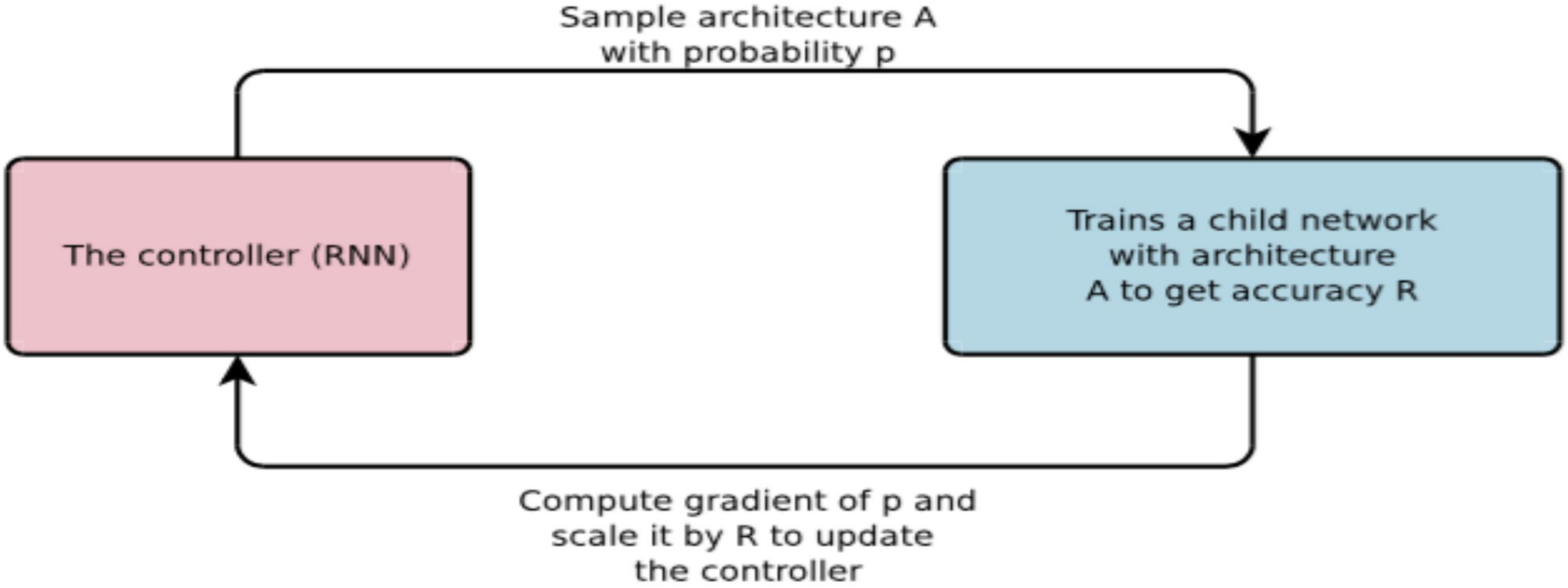
- Example:
- Initial positive review
- *Review starts out positive*
  At 117: "*not exactly a bad story*"
  At 158: "*I recommend this movie to everyone, even if you've never played the game*"

Neural Architecture Search

Neural Architecture Search!

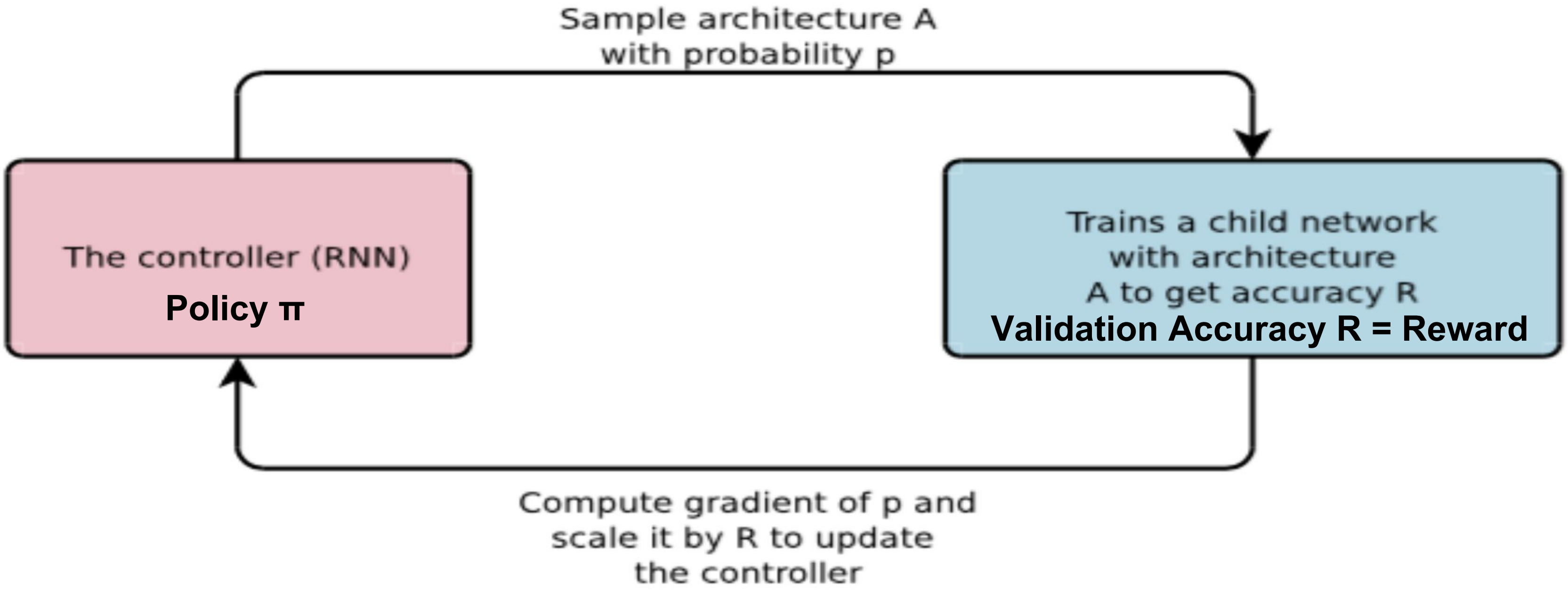
- Manual process of finding best units requires a lot of expertise
- What if we could use AI to find the right architecture for any problem?
- Neural architecture search with reinforcement learning by Zoph and Le, 2016

Neural Architecture Search



But R isn't differentiable

Neural Architecture Search



The trick : Reinforce

function REINFORCE

Initialise θ arbitrarily

for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

for $t = 1$ to $T - 1$ **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

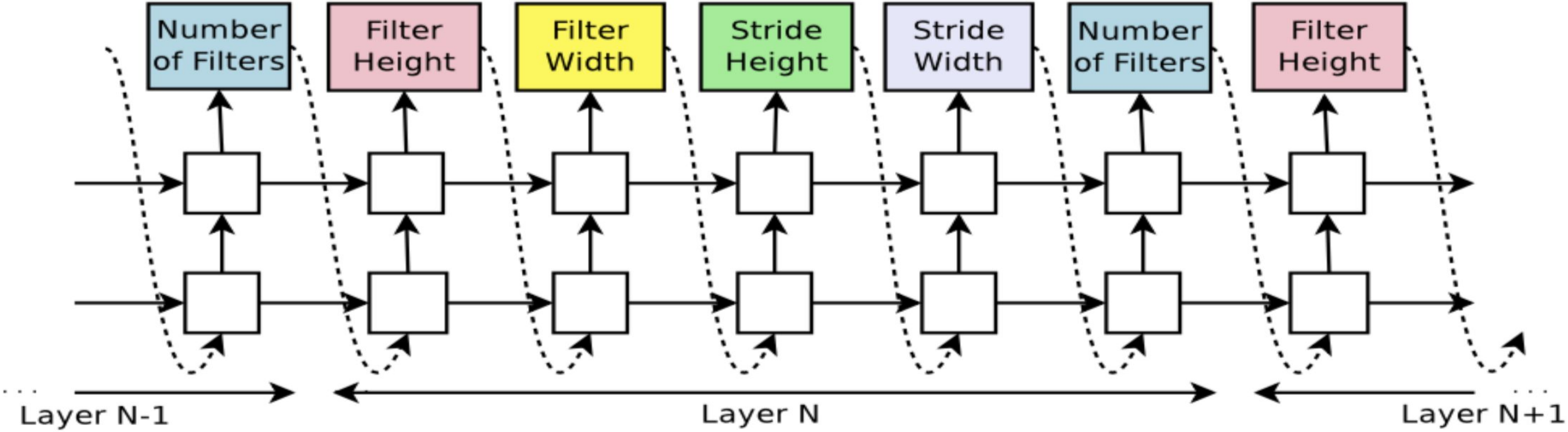
end for

end for

return θ

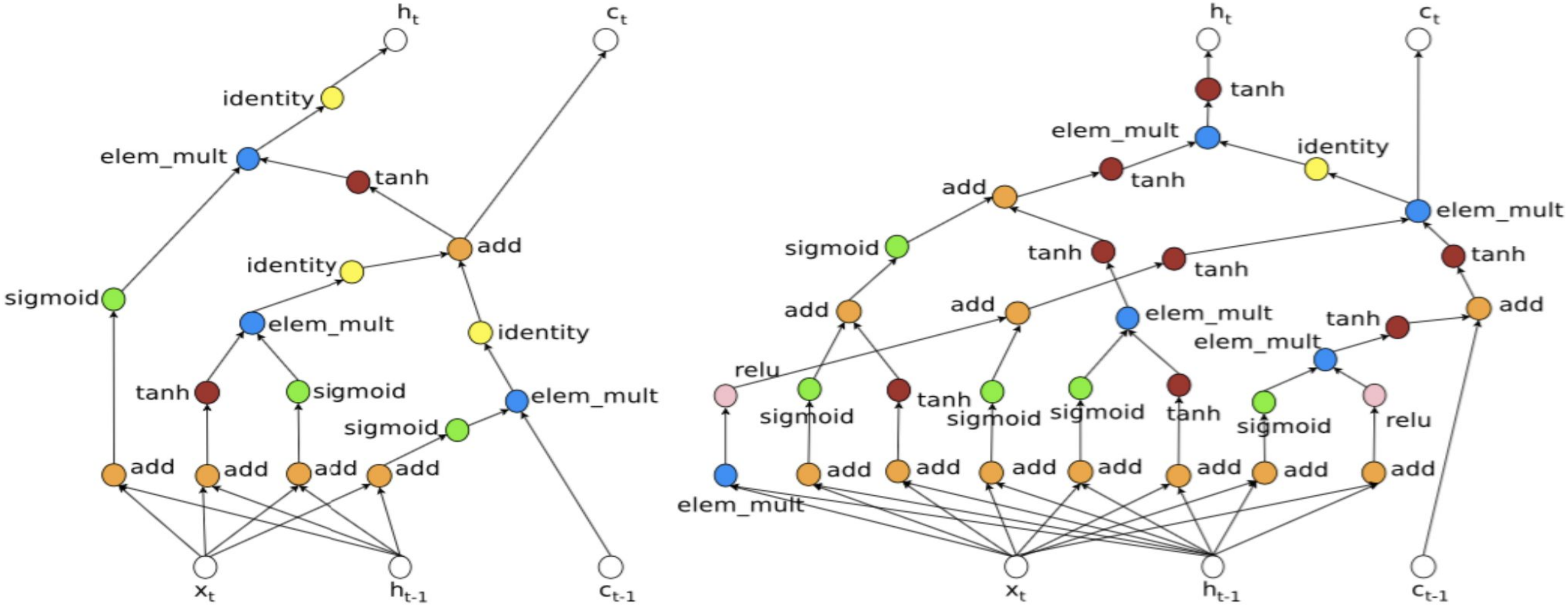
end function

Example: CNN Controller

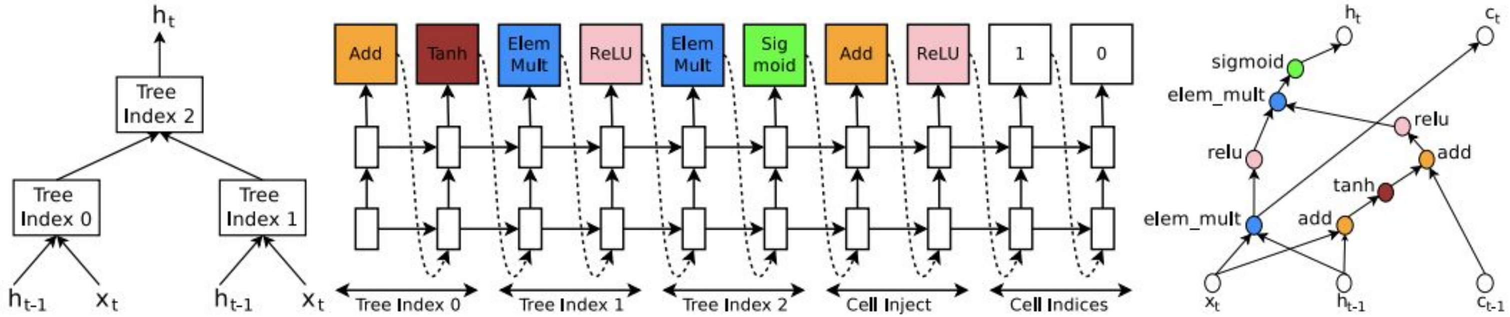


Used Reinforcement Learning to train the RNN Controller

LSTM Cell vs NAS Cell



Generating RNN units

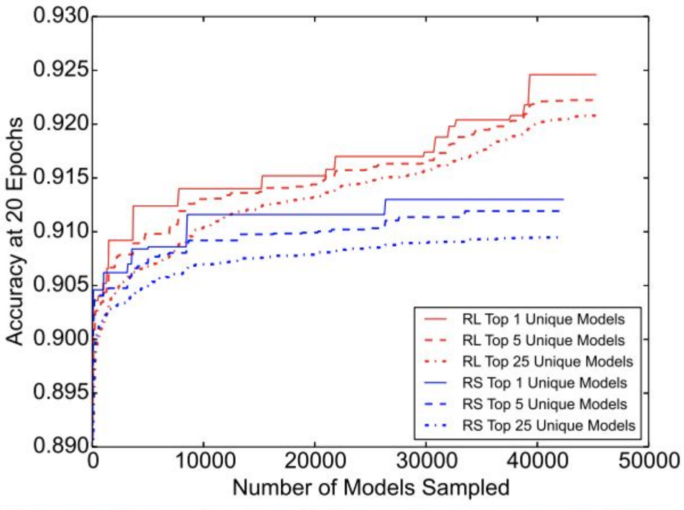


Nice Perplexity Reduction for Language Modeling

Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	51M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Inan et al. (2016) - VD-LSTM + REAL (large)	51M	68.5
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	66.0
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	62.4

But it takes 800 GPUs

Comparing NAS to random grid search



Soruce : Zoph, Barret, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. "Learning Transferable Architectures for Scalable Image Recognition." *arXiv preprint arXiv:1707.07012* (2017).

A better trick : PPO

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ

for $k = 0, 1, 2, \ldots$ **do**

Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Compute policy update

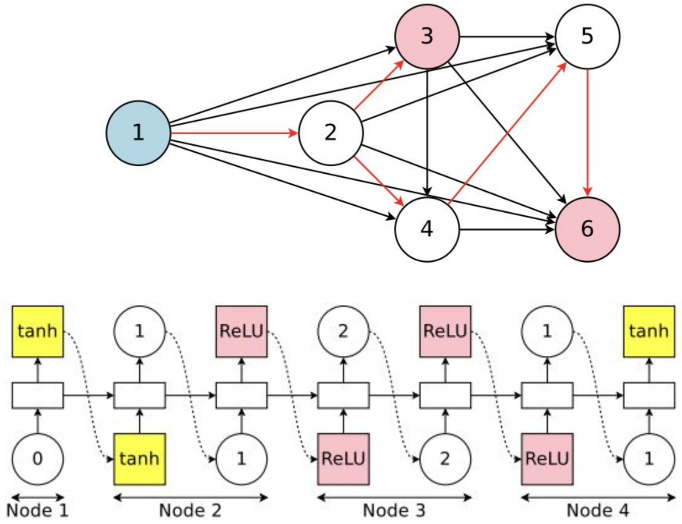
$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

end for

Sharing parameters



“Importantly, in all of our experiments, for which we use a single Nvidia GTX 1080Ti GPU, the search for architectures takes less than 16 hours”

Soruce : Pham, Hieu, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. "Efficient Neural Architecture Search via Parameter Sharing." *arXiv preprint arXiv:1802.03268* (2018).

AdaNet

The objective function

$$F(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\Phi\Big(1 - y_i\sum_{j=1}^{N} w_j h_j\Big) + \sum_{j=1}^{N}\Gamma_j |w_j|, \quad (4)$$

Let $x \mapsto \Phi(-x)$ be a non-increasing convex function upper-bounding the zero-one loss, $x \mapsto 1_{x \leq 0}$, such that Φ is differentiable over $\mathbb{R}$ and $\Phi'(x) \neq 0$ for all x.

Repo : https://github.com/tensorflow/adanet

$\text{AdaNet}(S = ((x_i, y_i)_{i=1}^{m})$

1 $f_0 \leftarrow 0$
2 **for** $t \leftarrow 1$ **to** T **do**
3 $\quad \mathbf{h}, \mathbf{h}' \leftarrow \text{WeakLearner}\big(S, f_{t-1}\big)$
4 $\quad \mathbf{w} \leftarrow \text{Minimize}\big(F_t(\mathbf{w}, \mathbf{h})\big)$
5 $\quad \mathbf{w}' \leftarrow \text{Minimize}\big(F_t(\mathbf{w}, \mathbf{h}')\big)$
6 $\quad$ **if** $F_t(\mathbf{w}, \mathbf{h}') \leq F_t(\mathbf{w}', \mathbf{h}')$ **then**
7 $\qquad \mathbf{h}_t \leftarrow \mathbf{h}$
8 $\quad$ **else** $\mathbf{h}_t \leftarrow \mathbf{h}'$
9 $\quad$ **if** $F(\mathbf{w}_{t-1} + \mathbf{w}^*) < F(\mathbf{w}_{t-1})$ **then**
10 $\qquad f_{t-1} \leftarrow f_t + \mathbf{w}^* \cdot \mathbf{h}_t$
11 $\quad$ **else** **return** f_{t-1}
12 **return** f_T

Figure 3. Pseudocode of the AdaNet algorithm. On line 3 two candidate subnetworks are generated (e.g. randomly or by solving (6)). On lines 3 and 4, (5) is solved for each of these candidates. On lines 5-7 the best subnetwork is selected and on lines 9-11 termination condition is checked.

$\min_{\mathbf{w}} F_t(\mathbf{w}, \mathbf{h}) \leq \min_{\mathbf{w}} F_t(\mathbf{w}, \mathbf{h}')$, then

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^B} F_t(\mathbf{w}, \mathbf{h}), \quad \mathbf{h}_t = \mathbf{h}$$

and otherwise

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^B} F_t(\mathbf{w}, \mathbf{h}'), \quad \mathbf{h}_t = \mathbf{h}'$$

Dynamic Memory Network

Dynamic Memory Network

- Motivation:

I:	Jane went to the hallway.
I:	Mary walked to the bathroom.
I:	Sandra went to the garden.
I:	Daniel went back to the garden.
I:	Sandra took the milk there.
Q:	Where is the milk?
A:	garden

Dynamic Memory Network

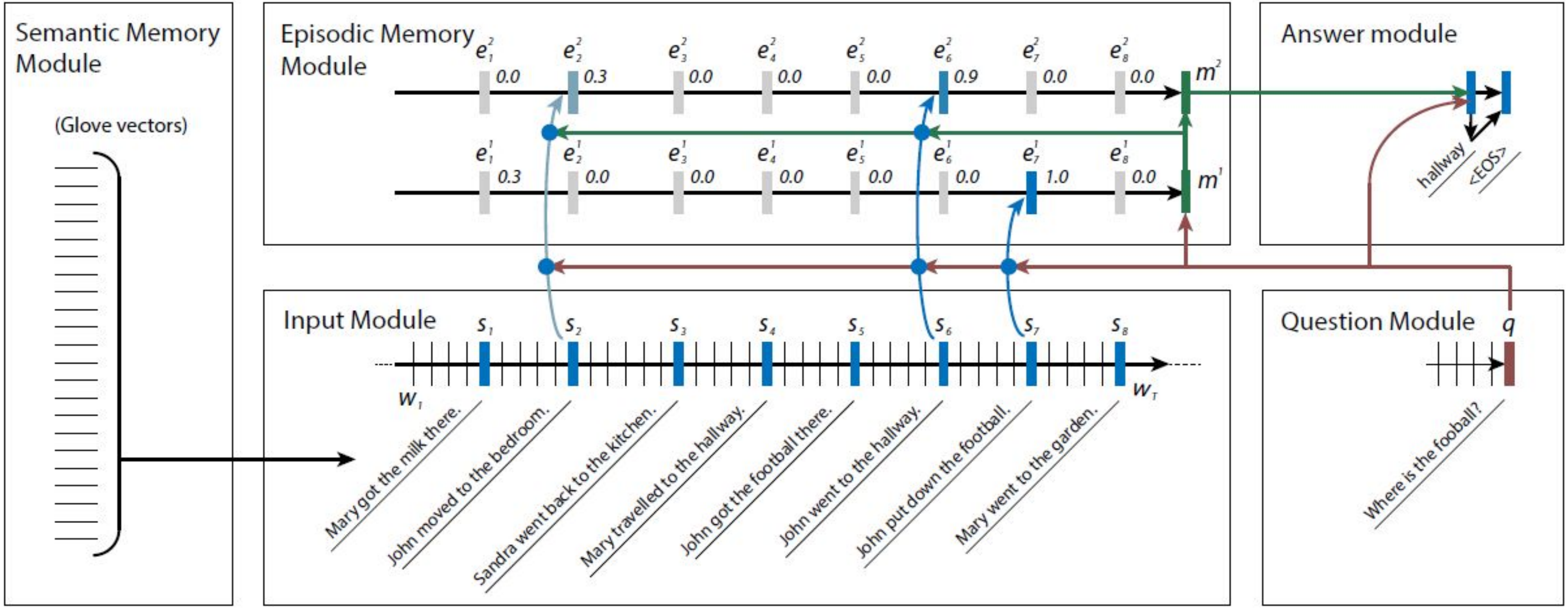
- Motivation:
  - Iterative attention process

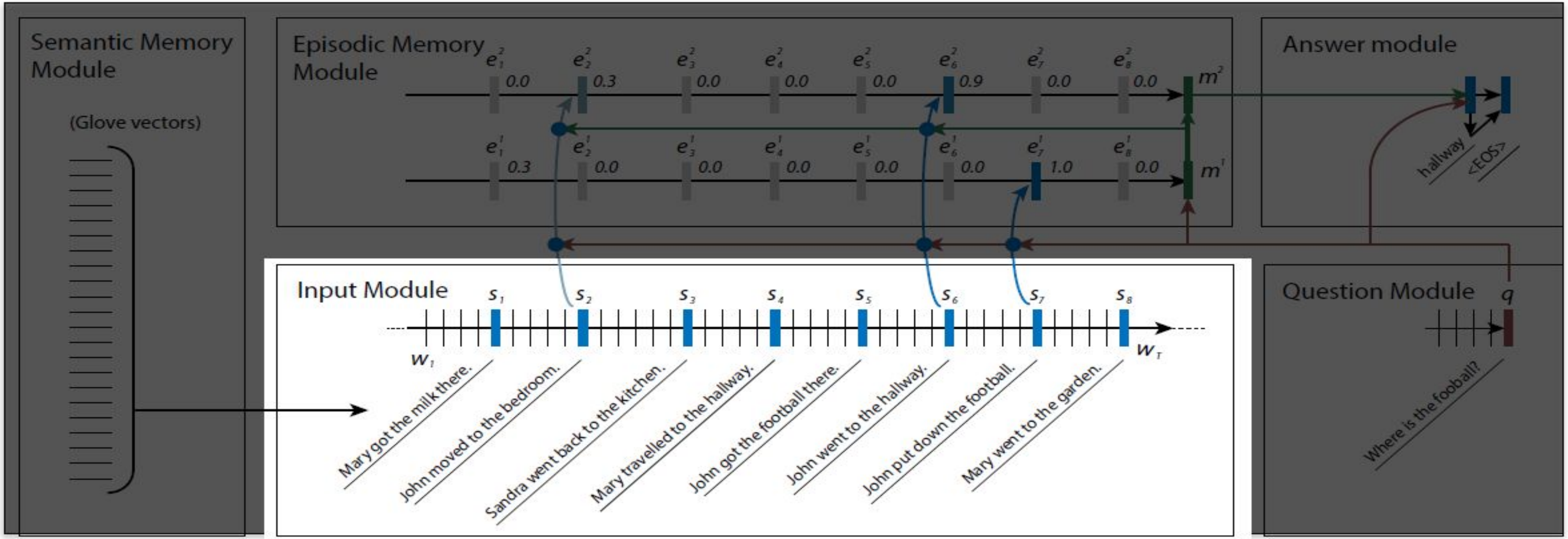
I: Jane went to the hallway.
I: Mary walked to the bathroom.
I: Sandra went to the garden.
I: Daniel went back to the garden.
I: Sandra took the milk there.
Q: Where is the milk?
A: garden

Initially we don’t pay attention on the sentence which contains the answers...

Since this iterative attention is good, need an architecture to strengthen that ability

Dynamic Memory Network



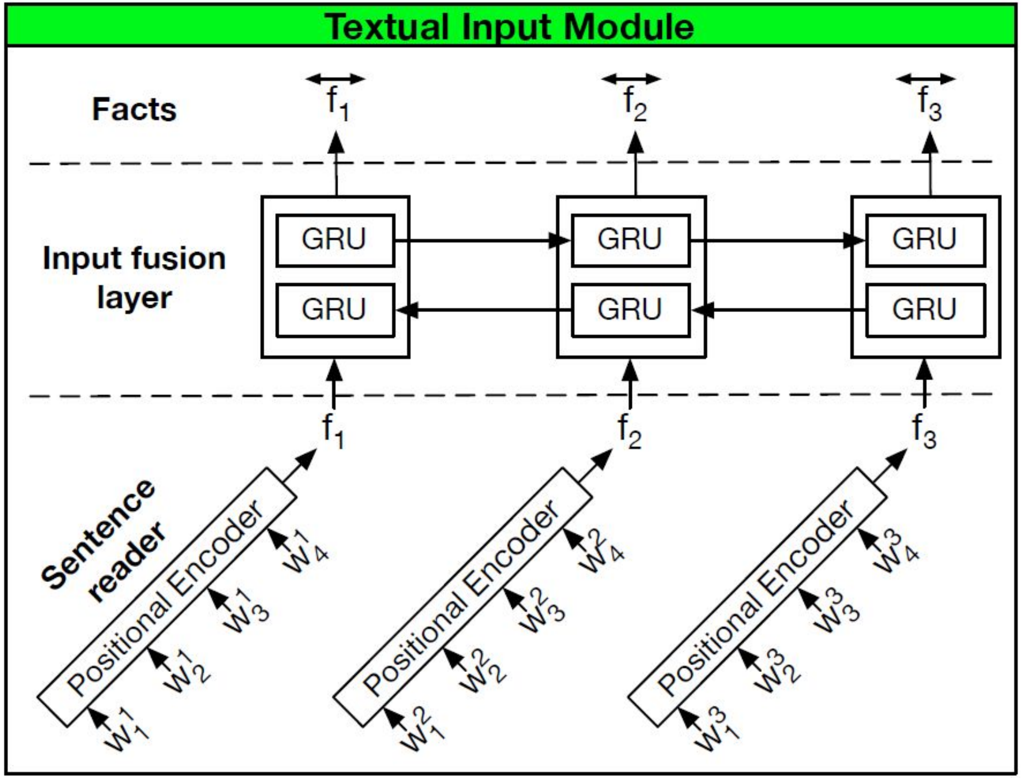


Standard GRU. The last hidden state of each sentence is accessible.

For one sentence input, keep hidden states of each words.

Optional: Having <End of Sentence> token and consider the hidden state of it as a representation of the sentence.

Further Improvement: BiGRU



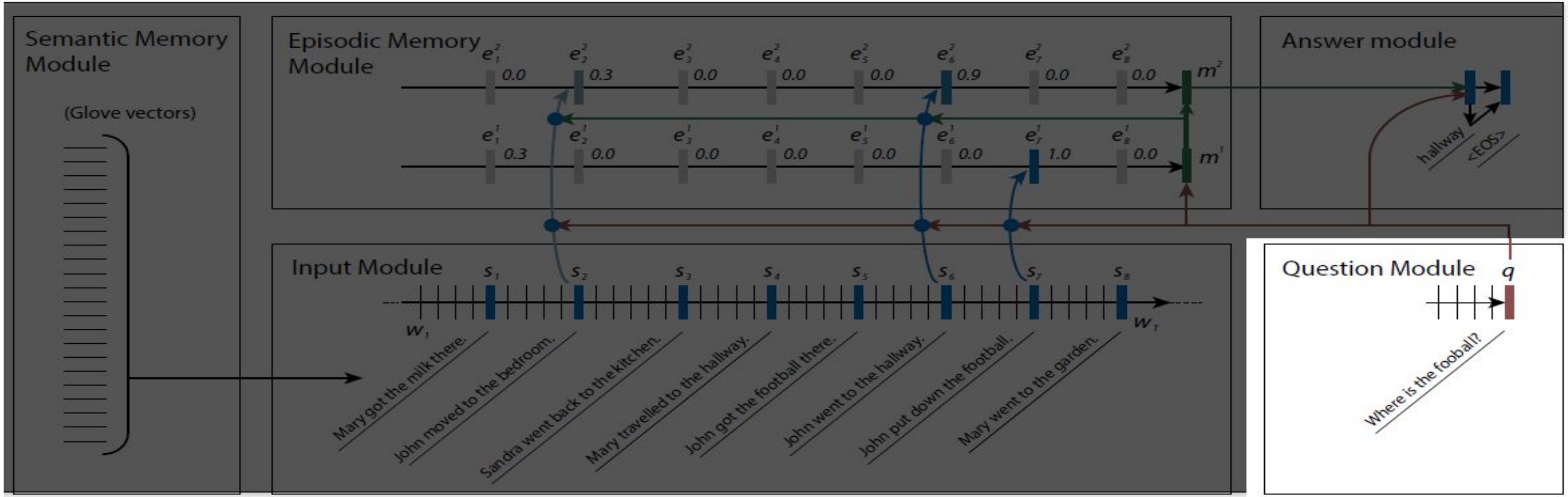
Position Encoding:

$$f_i = \sum_M^{j=1} l_j \circ w_j^i$$

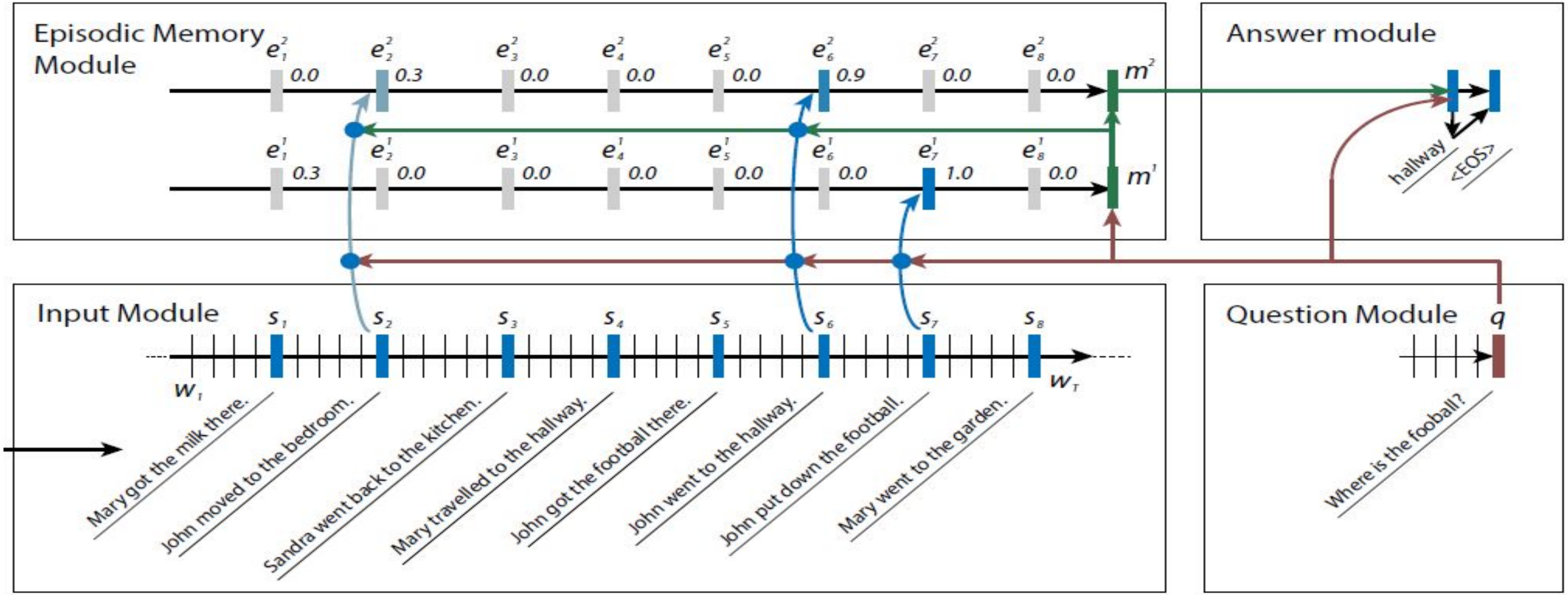
$$l_{jd} = (1 - j/M) - (d/D)(1 - 2j/M)$$

D: representation dimension
j: jth word in sentence
M: num word in sentence

The Modules: Question



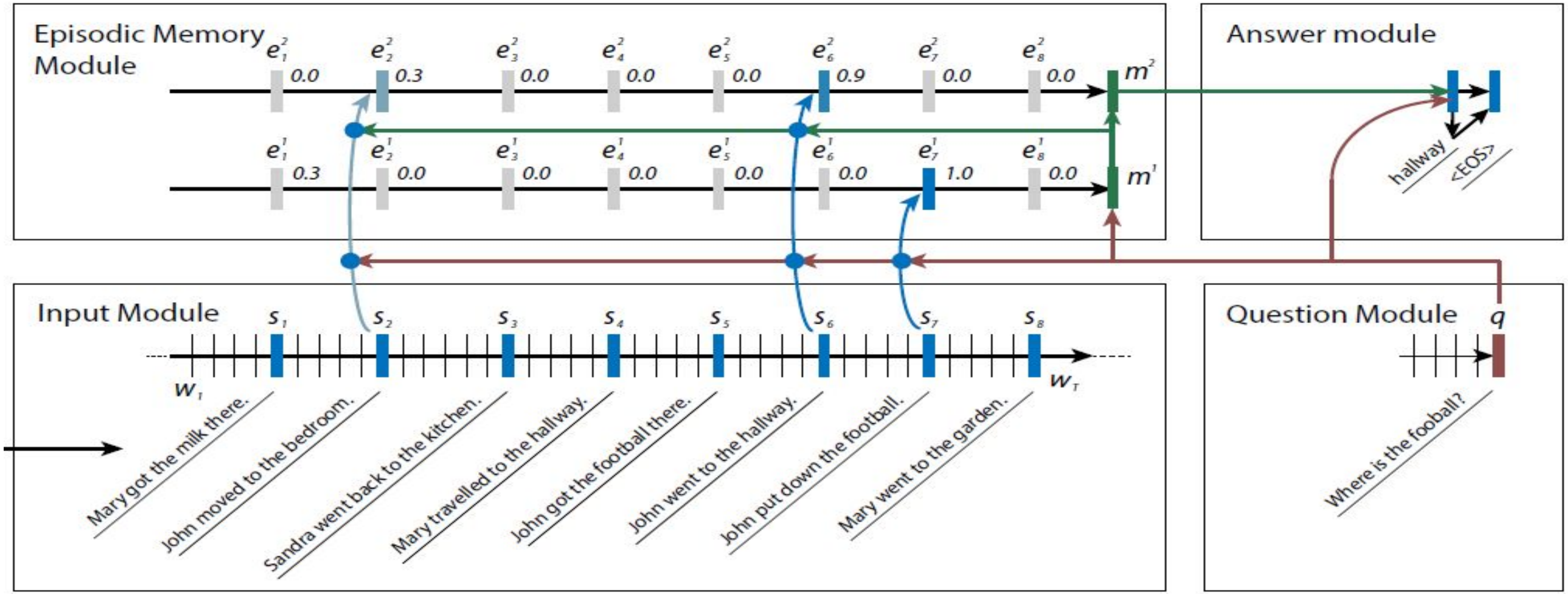
$$q_t = GRU(v_t, q_{t-1})$$



m_0 is initialized as q

At episode iteration t:

1. Using previous memory m_(t-1) and q to compute attention score:
   - similarity measure: z =[some features]-> .. -> softmax -> gi for each sentence,
1. Using a modified GRU with input (s1,s2 ,...) and (g1,g2,...) => last hidden state = m_t



$$z_i^t = [s_i \circ q \,; s_i \circ m^{t-1}; |s_i - q| \,; |s_i - m^{t-1}|]$$

$$h_i^t = g_i^t GRU\left(s_i, h_{i-1}^t\right) + \left(1 - g_i^t\right) h_{i-1}^t$$

$$Z_i^t = W^{(2)} \tanh\left(W^{(1)} z_i^t + b^{(1)}\right) + b^{(2)}$$

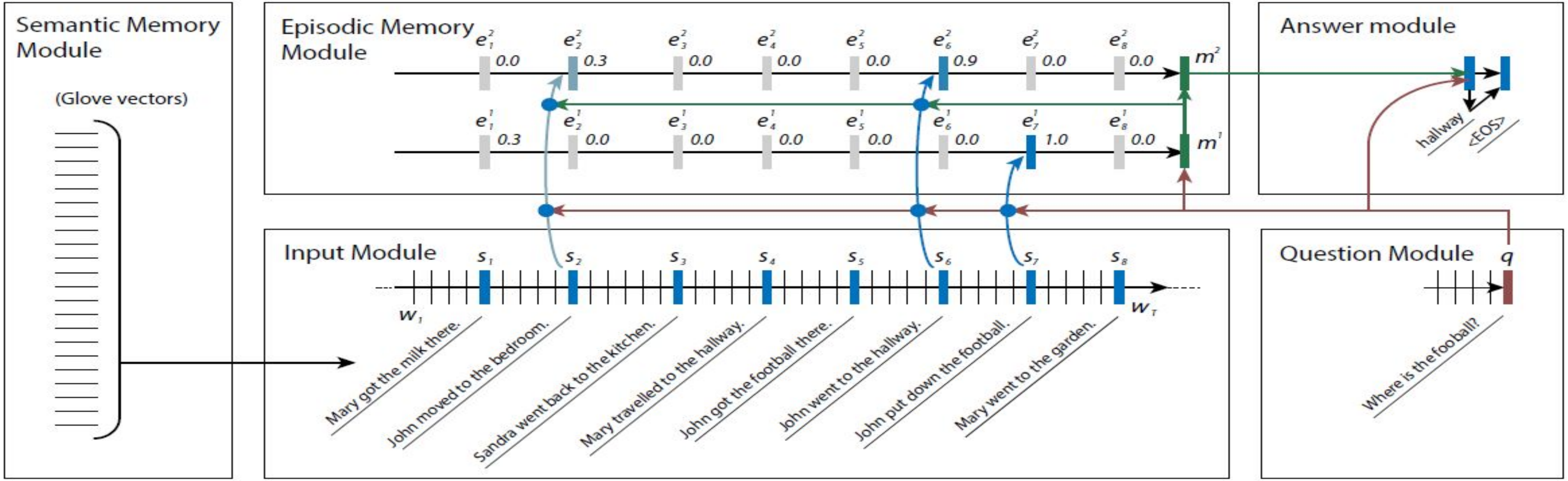
Last hidden state: m^t

$$g_i^t = \frac{\exp(Z_i^t)}{\sum_{k=1}^{M_i} \exp(Z_k^t)}$$

The Modules: Answer

$$a_t = \text{GRU}([y_{t-1}, q], a_{t-1}), \qquad y_t = softmax(W^{(a)} a_t)$$

Upgrade: using pointer to point to the input



Related work

- Sequence to Sequence (Sutskever et al. 2014)
- Neural Turing Machines (Graves et al. 2014)
- Teaching Machines to Read and Comprehend (Hermann et al. 2015)
- Learning to Transduce with Unbounded Memory (Grefenstette 2015)
- Structured Memory for Neural Turing Machines (Wei Zhang 2015)

- Memory Networks (Weston et al. 2015)
- End to end memory networks (Sukhbaatar et al. 2015)
  →

Memory networks

Array of memory m

I: (input feature map) – converts the incoming input to the internal feature representation.

Using features (embedding, POS,coreference,position encoding,...)

G: (generalization) – updates old memories given the new input. We call this generalization as there is an opportunity for the network to compress and generalize its memories at this stage for some intended future use. Could be simple as putting new element to m

O: (output feature map) – produces a new output (in the feature representation space), given the new input and the current memory state.

The O component is typically responsible for reading from memory and performing inference. Have some function for scoring attention.
Similar with DMN, produce output 1 and condition on it and memory to find output 2 and so on… -> final output

R: (response) – converts the output into the response format desired. For example, a textual response or an action.

Memory Networks (Weston et al. 2015)

Comparison to MemNets

Similarities:

- MemNets and DMNs have input, scoring, attention and response mechanisms

Differences:

- For input representations MemNets use bag of word, nonlinear or linear embeddings that explicitly encode position
- MemNets iteratively run functions for attention and response

- **DMNs show that neural sequence models can be used for input representation, attention and response mechanisms**
  → naturally captures position and temporality
- Enables broader range of applications

babI 1k, with gate supervision

Task 2,3 have long input sequence, DMN do poor while MemNN because it views each sentence separately.
Taks 7,8 requires iteratively retrieve facts and slowly incorporate, DNN shows better result.

Task	MemNN	DMN	Task	MemNN	DMN
1: Single Supporting Fact	100	100	11: Basic Coreference	100	99.9
2: Two Supporting Facts	100	98.2	12: Conjunction	100	100
3: Three Supporting facts	100	95.2	13: Compound Coreference	100	99.8
4: Two Argument Relations	100	100	14: Time Reasoning	99	100
5: Three Argument Relations	98	99.3	15: Basic Deduction	100	100
6: Yes/No Questions	100	100	16: Basic Induction	100	99.4
7: Counting	85	96.9	17: Positional Reasoning	65	59.6
8: Lists/Sets	91	96.5	18: Size Reasoning	95	95.3
9: Simple Negation	100	100	19: Path Finding	36	34.5
10: Indefinite Knowledge	98	97.5	20: Agent's Motivations	100	100
			Mean Accuracy (%)	93.3	**93.6**

Experiments: Sentiment Analysis

Stanford Sentiment Treebank

Test accuracies:

- MV-RNN and RNTN: Socher et al. (2013)
- DCNN: Kalchbrenner et al. (2014)
- PVec: Le & Mikolov. (2014)
- CNN-MC: Kim (2014)
- DRNN: Irsoy & Cardie (2015)
- CT-LSTM: Tai et al. (2015)

Task	Binary	Fine-grained
MV-RNN	82.9	44.4
RNTN	85.4	45.7
DCNN	86.8	48.5
PVec	87.8	48.7
CNN-MC	88.1	47.4
DRNN	86.6	49.8
CT-LSTM	88.0	51.0
DMN	**88.6**	**52.1**

Analysis of Number of Episodes

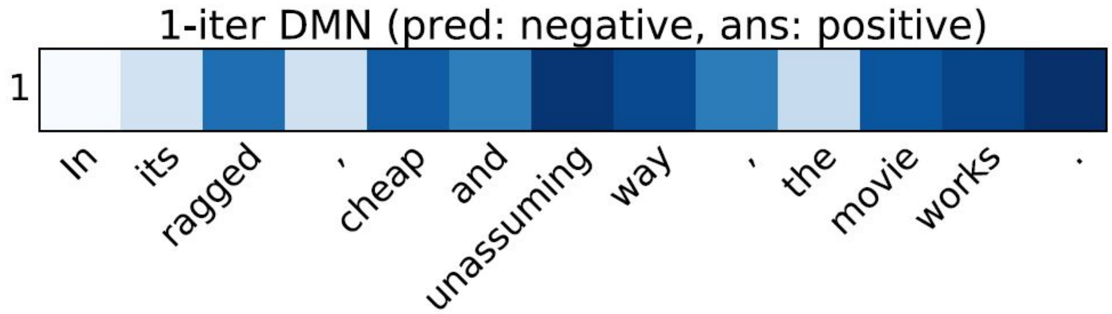
- How many attention + memory passes are needed in the episodic memory?

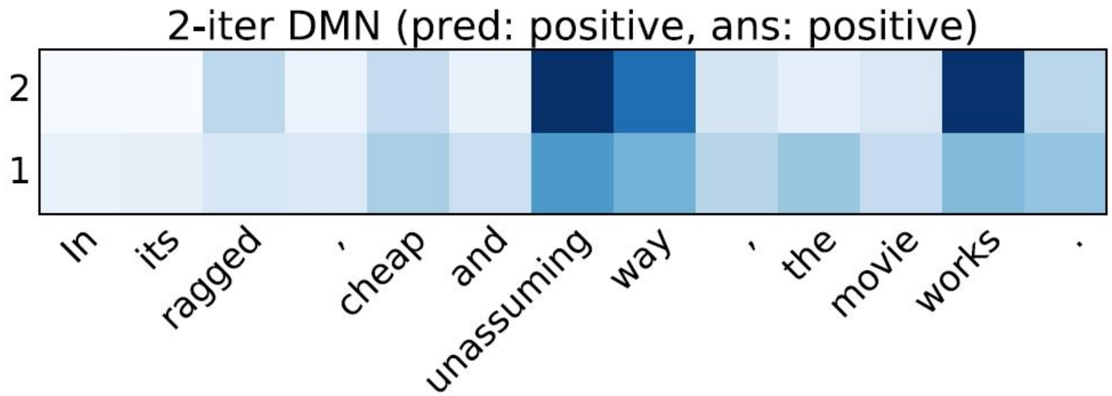
Max passes	task 3 three-facts	task 7 count	task 8 lists/sets	sentiment (fine grain)
0 pass	0	48.8	33.6	50.0
1 pass	0	48.8	54.0	51.5
2 pass	16.7	49.1	55.6	**52.1**
3 pass	64.7	83.4	83.4	50.1
5 pass	**95.2**	**96.9**	**96.5**	N/A

Should consider the number of passes as hyperparameter.

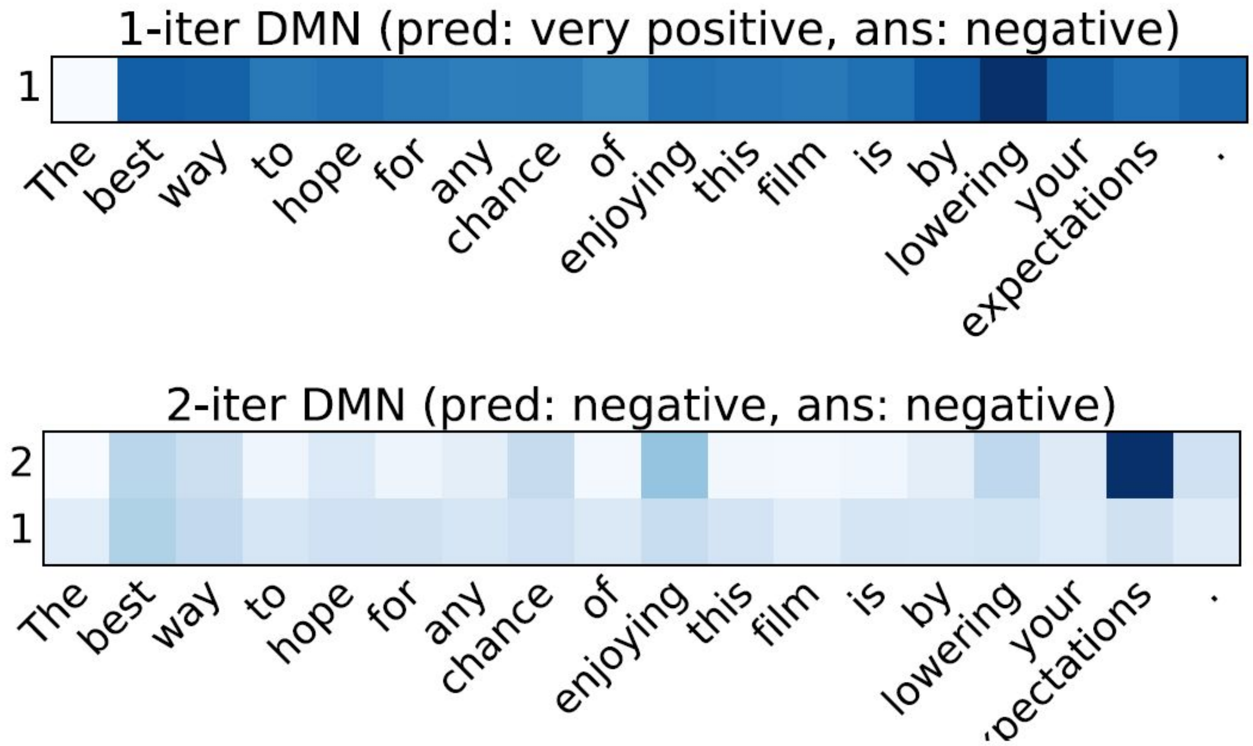
Analysis of Attention for Sentiment

- Sharper attention when 2 passes are allowed.
- Examples that are wrong with just one pass





Analysis of Attention for Sentiment

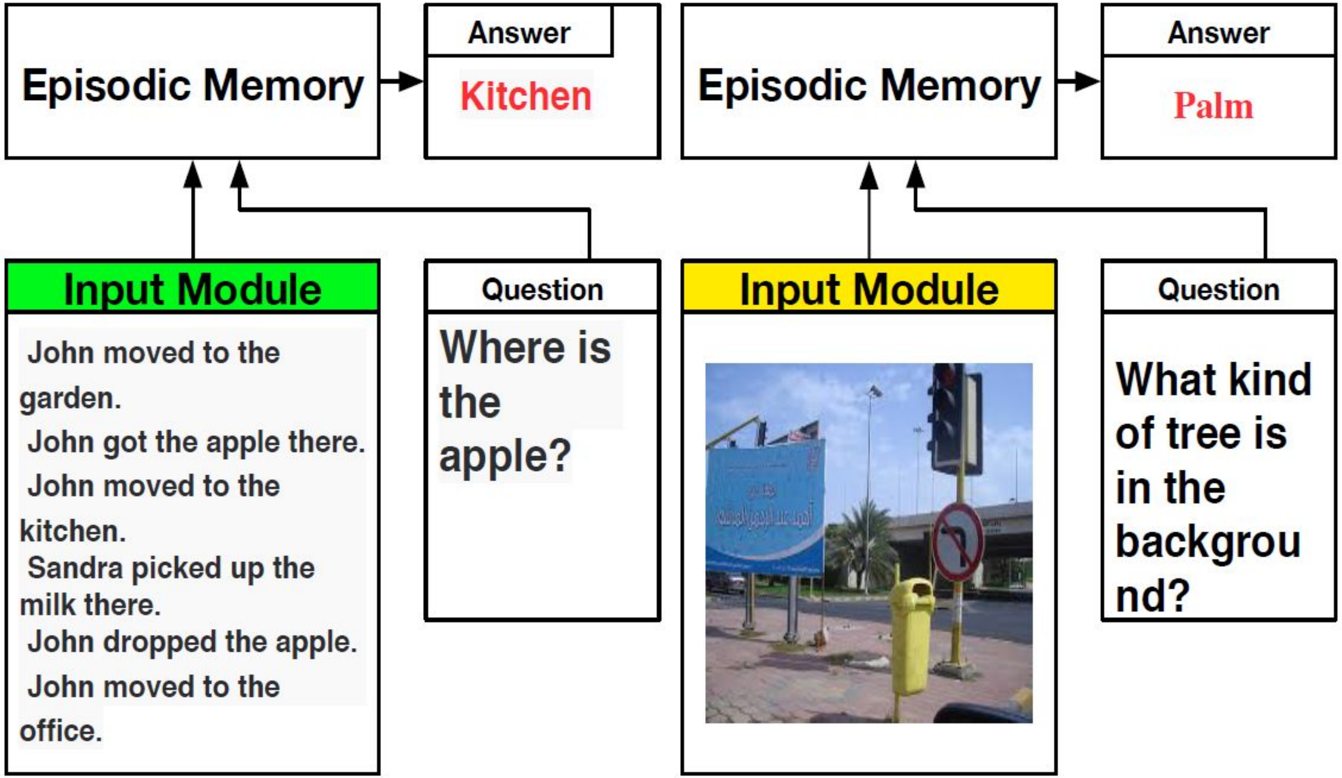


Experiments: POS Tagging

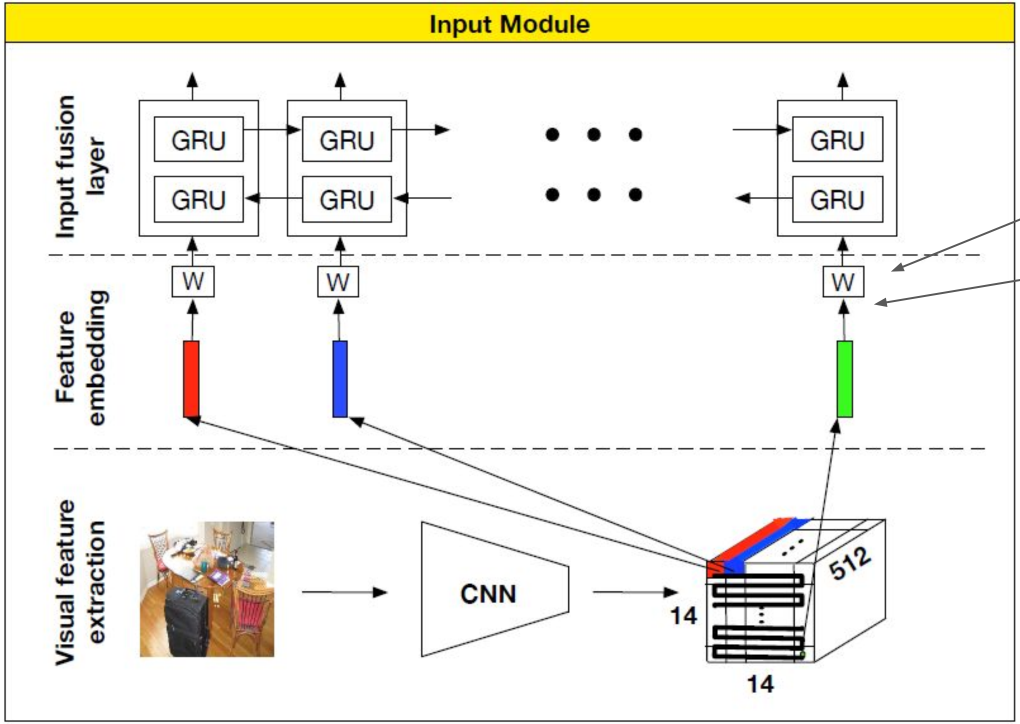
- PTB WSJ, standard splits
- Episodic memory does not require multiple passes, single pass enough

Model	SVMTool	Sogaard	Suzuki et al.	Spoustova et al.	SCNN	DMN
Acc (%)	97.15	97.27	97.40	97.44	97.50	**97.56**

Modularization Allows for Different Inputs



Input Module for Images



Representation of a feature

Linear layer with tanh activation

Project to Visual feature embedding

Local region feature extraction (feature blocks)

Dynamic Memory Networks for Visual and Textual Question Answering, Caiming Xiong, Stephen Merity, Richard Socher

Accuracy: Visual Question Answering

VQA test-dev and test-standard:

- Antol et al. (2015)
- ACK Wu et al. (2015);
- iBOWIMG - Zhou et al. (2015);
- DPPnet - Noh et al. (2015); D-NMN - Andreas et al. (2016);
- SAN - Yang et al. (2015)

	test-dev				test-std
Method	All	Y/N	Other	Num	All
VQA					
Image	28.1	64.0	3.8	0.4	-
Question	48.1	75.7	27.1	36.7	-
Q+I	52.6	75.6	37.4	33.7	-
LSTM Q+I	53.7	78.9	36.4	35.2	54.1
ACK	55.7	79.2	40.1	36.1	56.0
iBOWIMG	55.7	76.5	42.6	35.0	55.9
DPPnet	57.2	80.7	41.7	37.2	57.4
D-NMN	57.9	80.5	43.1	37.4	58.0
SAN	58.7	79.3	46.1	36.6	58.9
DMN+	**60.3**	80.5	48.3	36.8	**60.4**

Attention Visualization

What is this sculpture made out of ?

Answer: metal

What color are the bananas ?

Answer: green

What is the pattern on the cat ' s fur on its tail ?

Answer: stripes

Did the player hit the ball ?

Answer: yes