Model Overview and Memory Networks

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

- 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.

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

Child-Sum Tree-LSTM

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

\[
i_j = \sigma \left( W^{(i)} x_j + U^{(i)} h_j + b^{(i)} \right), \quad (3)
\]

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

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

\[
u_{j} = \tanh \left( W^{(a)} x_j + U^{(a)} \tilde{h}_j + b^{(a)} \right), \quad (6)
\]

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

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

- Hidden state is sum of all child nodes’ hidden states
- Uses the same weight matrix for all child nodes
- Memory from child nodes is the sum of all child hidden states multiplied by forget gate values

- 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

\[ i_j = \sigma \left( W^{(i)} x_j + \sum_{t=1}^{N} U^{(i)}_{t} h_{jt} + b^{(i)} \right), \quad (9) \]

\[ f_j = \sigma \left( W^{(f)} x_j + \sum_{t=1}^{N} U^{(f)}_{t} h_{jt} + b^{(f)} \right), \quad (10) \]

\[ o_j = \sigma \left( W^{(o)} x_j + \sum_{t=1}^{N} U^{(o)}_{t} h_{jt} + b^{(o)} \right), \quad (11) \]

\[ u_j = \tanh \left( W^{(u)} x_j + \sum_{t=1}^{N} U^{(u)}_{t} h_{jt} + b^{(u)} \right), \quad (12) \]

\[ c_j = i_j \odot u_j + \sum_{t=1}^{N} f_j \odot c_{jt}, \quad (13) \]

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

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
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)

\[ z_t = \tanh(W^1 x_{t-1} + W^2 x_t) \]
\[ f_t = \sigma(W^1 x_{t-1} + W^2 x_t) \]
\[ o_t = \sigma(W^1 x_{t-1} + W^2 x_t) \]

- 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

![QRNN Diagram]

**Pooling Layer**

\[
\begin{align*}
h_t &= f_t \odot h_{t-1} + (1 - f_t) \odot z_t, \\
c_t &= f_t \odot c_{t-1} + (1 - f_t) \odot z_t \\
h_t &= o_t \odot c_t.
\end{align*}
\]

- **f-Pooling**
- **fo-Pooling**
- **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

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameters</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM (medium) (Zaremba et al., 2014)</td>
<td>20M</td>
<td>86.2</td>
<td>82.7</td>
</tr>
<tr>
<td>Variational LSTM (medium) (Gal &amp; Ghahramani, 2016)</td>
<td>20M</td>
<td>81.9</td>
<td>79.7</td>
</tr>
<tr>
<td>LSTM with CharCNN embeddings (Kim et al., 2016)</td>
<td>19M</td>
<td>-</td>
<td>78.9</td>
</tr>
<tr>
<td>Zoneout + Variational LSTM (medium) (Merity et al., 2016)</td>
<td>20M</td>
<td>84.4</td>
<td>80.6</td>
</tr>
<tr>
<td><strong>Our models</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTM (medium)</td>
<td>20M</td>
<td>85.7</td>
<td>82.0</td>
</tr>
<tr>
<td>QRNN (medium)</td>
<td>18M</td>
<td>82.9</td>
<td>79.9</td>
</tr>
<tr>
<td>QRNN + zoneout ($p = 0.1$) (medium)</td>
<td>18M</td>
<td>82.1</td>
<td>78.3</td>
</tr>
</tbody>
</table>

- Faster

![Graph showing RNN, Softmax, and Optimization Overhead times](image)

<table>
<thead>
<tr>
<th>Batch size</th>
<th>Sequence length</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>5.5x 8.8x 11.0x 12.4x 16.9x</td>
</tr>
<tr>
<td>64</td>
<td>5.5x 6.7x 7.8x 8.3x 10.8x</td>
</tr>
<tr>
<td>128</td>
<td>4.2x 4.5x 4.9x 4.9x 6.4x</td>
</tr>
<tr>
<td>256</td>
<td>3.0x 3.0x 3.0x 3.0x 3.7x</td>
</tr>
<tr>
<td>512</td>
<td>1.4x 1.4x 1.3x 1.3x 1.3x</td>
</tr>
</tbody>
</table>
Q-RNNs: Sentiment Analysis

- Often better and faster than LSTMs
- 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!

- 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

1. Sample architecture A with probability p
2. The controller (RNN)
3. Trains a child network with architecture A to get accuracy R
4. Compute gradient of p and scale it by R to update the controller
But R isn’t differentiable
Neural Architecture Search

Sample architecture A with probability $p$

The controller (RNN)

Policy $\pi$

Trains a child network with architecture A to get accuracy $R$

Validation Accuracy $R = \text{Reward}$

Compute gradient of $p$ and scale it by $R$ to update the controller
The trick: Reinforce

function REINFORCE
    Initialise $\theta$ 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 $\theta$
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

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameters</th>
<th>Test Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mikolov &amp; Zweig (2012) - KN-5</td>
<td>2M‡</td>
<td>141.2</td>
</tr>
<tr>
<td>Mikolov &amp; Zweig (2012) - KN5 + cache</td>
<td>2M‡</td>
<td>125.7</td>
</tr>
<tr>
<td>Mikolov &amp; Zweig (2012) - RNN</td>
<td>6M‡</td>
<td>124.7</td>
</tr>
<tr>
<td>Mikolov &amp; Zweig (2012) - RNN-LDA</td>
<td>7M‡</td>
<td>113.7</td>
</tr>
<tr>
<td>Mikolov &amp; Zweig (2012) - RNN-LDA + KN-5 + cache</td>
<td>9M‡</td>
<td>92.0</td>
</tr>
<tr>
<td>Pascanu et al. (2013) - Deep RNN</td>
<td>6M</td>
<td>107.5</td>
</tr>
<tr>
<td>Cheng et al. (2014) - Sum-Prod Net</td>
<td>5M‡</td>
<td>100.0</td>
</tr>
<tr>
<td>Zaremba et al. (2014) - LSTM (medium)</td>
<td>20M</td>
<td>82.7</td>
</tr>
<tr>
<td>Zaremba et al. (2014) - LSTM (large)</td>
<td>66M</td>
<td>78.4</td>
</tr>
<tr>
<td>Gal (2015) - Variational LSTM (medium, untied)</td>
<td>20M</td>
<td>79.7</td>
</tr>
<tr>
<td>Gal (2015) - Variational LSTM (medium, untied, MC)</td>
<td>20M</td>
<td>78.6</td>
</tr>
<tr>
<td>Gal (2015) - Variational LSTM (large, untied)</td>
<td>66M</td>
<td>75.2</td>
</tr>
<tr>
<td>Gal (2015) - Variational LSTM (large, untied, MC)</td>
<td>66M</td>
<td>73.4</td>
</tr>
<tr>
<td>Kim et al. (2015) - CharCNN</td>
<td>19M</td>
<td>78.9</td>
</tr>
<tr>
<td>Press &amp; Wolf (2016) - Variational LSTM, shared embeddings</td>
<td>51M</td>
<td>73.2</td>
</tr>
<tr>
<td>Merity et al. (2016) - Zoneout + Variational LSTM (medium)</td>
<td>20M</td>
<td>80.6</td>
</tr>
<tr>
<td>Merity et al. (2016) - Pointer Sentinel-LSTM (medium)</td>
<td>21M</td>
<td>70.9</td>
</tr>
<tr>
<td>Inan et al. (2016) - VD-LSTM + REAL (large)</td>
<td>51M</td>
<td>68.5</td>
</tr>
<tr>
<td>Zilly et al. (2016) - Variational RHN, shared embeddings</td>
<td>24M</td>
<td>66.0</td>
</tr>
<tr>
<td>Neural Architecture Search with base 8</td>
<td>32M</td>
<td>67.9</td>
</tr>
<tr>
<td>Neural Architecture Search with base 8 and shared embeddings</td>
<td>25M</td>
<td>64.0</td>
</tr>
<tr>
<td>Neural Architecture Search with base 8 and shared embeddings</td>
<td>54M</td>
<td>62.4</td>
</tr>
</tbody>
</table>
But it takes 800 GPUs
Comparing NAS to random grid search

A better trick: PPO

Algorithm 5 PPO with Clipped Objective

\begin{algorithm}
\begin{algorithmic}
  \STATE Input: initial policy parameters $\theta_0$, clipping threshold $\epsilon$
  \FOR {$k = 0, 1, 2, \ldots$}
    \STATE Collect set of partial trajectories $D_k$ on policy $\pi_k = \pi(\theta_k)$
    \STATE Estimate advantages $\hat{A}^\pi_{t}$ using any advantage estimation algorithm
    \STATE Compute policy update
    \begin{equation}
    \theta_{k+1} = \arg\max_{\theta} L^{CLIP}_{\theta_k}(\theta)
    \end{equation}
    \STATE by taking $K$ steps of minibatch SGD (via Adam), where
    \begin{equation}
    L^{CLIP}_{\theta_k}(\theta) = \mathbb{E}_{\tau \sim \pi_k} \left[ \sum_{t=0}^{T} \min(\tau_t(\theta)\hat{A}^\pi_{t}, \text{clip}(\tau_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^\pi_{t}) \right]
    \end{equation}
  \ENDFOR
\end{algorithmic}
\end{algorithm}
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

AdaNet

The objective function

\[ F(w) = \frac{1}{m} \sum_{i=1}^{m} \Phi \left( 1 - y_i \sum_{j=1}^{N} w_j h_j \right) + \sum_{j=1}^{N} \Gamma_j |w_j|, \quad (4) \]

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 \( \Phi \) is differentiable over \( \mathbb{R} \) and \( \Phi'(x) \neq 0 \) for all \( x \).

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

---

**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. \( \text{h, h'} \leftarrow \text{WEAKLEARNER}(S, f_{t-1}) \)
4. \( w \leftarrow \text{MINIMIZE}(F_t(w, h)) \)
5. \( w' \leftarrow \text{MINIMIZE}(F_t(w, h')) \)
6. if \( F_t(w, h') \leq F_t(w', h') \) then
7. \( h_t \leftarrow h \)
8. else \( h_t \leftarrow h' \)
9. if \( F(w_{t-1} + w^*) < F(w_{t-1}) \) then
10. \( f_{t-1} \leftarrow f_t + w^* \cdot h_t \)
11. 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.

\[ \min_w F_t(w, h) \leq \min_w F_t(w, h'), \text{then} \]

\[ w^* = \arg\min_{w \in \mathbb{R}^B} F_t(w, h), \quad h_t = h \]

and otherwise

\[ w^* = \arg\min_{w \in \mathbb{R}^B} F_t(w, h'), \quad h_t = h' \]
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
Motivation:
  ○ Iterative attention process

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
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:

\[ f_i = \sum_{j=1}^{M} l_j \circ w_j^{i} \]

\[ l_{jd} = (1 - j/M) - (d/D)(1 - 2j/M) \]

D: representation dimension
j: jth word in sentence
M: num word in sentence
\[ q_t = GRU(v_t, q_{t-1}) \]
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: $z = [\text{some features}] \rightarrow \ldots \rightarrow \text{softmax} \rightarrow g_i$ for each sentence,
2. Using a modified GRU with input (s1, s2, ...) and (g1, g2, ...) => last hidden state = m_t
$h_i^t = g_i^t \text{GRU}(s_i, h_{i-1}^t) + (1 - g_i^t)h_{i-1}^t$

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

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

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

Last hidden state: $m^t$
The Modules: Answer

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

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

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

|                                    |       |      |
|                                    | 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)
- CT-LSTM: Tai et al. (2015)

<table>
<thead>
<tr>
<th>Task</th>
<th>Binary</th>
<th>Fine-grained</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV-RNN</td>
<td>82.9</td>
<td>44.4</td>
</tr>
<tr>
<td>RNTN</td>
<td>85.4</td>
<td>45.7</td>
</tr>
<tr>
<td>DCNN</td>
<td>86.8</td>
<td>48.5</td>
</tr>
<tr>
<td>PVec</td>
<td>87.8</td>
<td>48.7</td>
</tr>
<tr>
<td>CNN-MC</td>
<td>88.1</td>
<td>47.4</td>
</tr>
<tr>
<td>DRNN</td>
<td>86.6</td>
<td>49.8</td>
</tr>
<tr>
<td>CT-LSTM</td>
<td>88.0</td>
<td>51.0</td>
</tr>
<tr>
<td>DMN</td>
<td><strong>88.6</strong></td>
<td><strong>52.1</strong></td>
</tr>
</tbody>
</table>
Analysis of Number of Episodes

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

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

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

1-iter DMN (pred: negative, ans: positive)

2-iter DMN (pred: positive, ans: positive)
Analysis of Attention for Sentiment

1-iter DMN (pred: very positive, ans: negative)

2-iter DMN (pred: negative, ans: negative)
Experiments: POS Tagging

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

<table>
<thead>
<tr>
<th>Model</th>
<th>SVMTool</th>
<th>Sogaard</th>
<th>Suzuki et al.</th>
<th>Spoustova et al.</th>
<th>SCNN</th>
<th>DMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc (%)</td>
<td>97.15</td>
<td>97.27</td>
<td>97.40</td>
<td>97.44</td>
<td>97.50</td>
<td><strong>97.56</strong></td>
</tr>
</tbody>
</table>
Modularization Allows for Different Inputs

Episodic Memory → Answer
Kitchen

Episodic Memory → Answer
Palm

Input Module
John moved to the garden.
John got the apple there.
John moved to the kitchen.
Sandra picked up the milk there.
John dropped the apple.
John moved to the office.

Question
Where is the apple?

Input Module

Question
What kind of tree is in the background?
Local region feature extraction (feature blocks)

Project to Visual feature embedding

Linear layer with tanh activation

Representation of a feature

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

<table>
<thead>
<tr>
<th>Method</th>
<th>test-dev</th>
<th>test-std</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>Y/N</td>
</tr>
<tr>
<td>VQA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image</td>
<td>28.1</td>
<td>64.0</td>
</tr>
<tr>
<td>Question</td>
<td>48.1</td>
<td>75.7</td>
</tr>
<tr>
<td>Q+I</td>
<td>52.6</td>
<td>75.6</td>
</tr>
<tr>
<td>LSTM Q+I</td>
<td>53.7</td>
<td>78.9</td>
</tr>
<tr>
<td>ACK</td>
<td>55.7</td>
<td>79.2</td>
</tr>
<tr>
<td>iBOWIMG</td>
<td>55.7</td>
<td>76.5</td>
</tr>
<tr>
<td>DPPnet</td>
<td>57.2</td>
<td>80.7</td>
</tr>
<tr>
<td>D-NMN</td>
<td>57.9</td>
<td>80.5</td>
</tr>
<tr>
<td>SAN</td>
<td>58.7</td>
<td>79.3</td>
</tr>
<tr>
<td>DMN+</td>
<td><strong>60.3</strong></td>
<td>80.5</td>
</tr>
</tbody>
</table>

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)
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