Inferring and Executing Programs for Visual Reasoning

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C.Lawrence Zitnick, Ross Girshick

Presenter: Siliang Lu
9/26/2017
What is visual reasoning?

- In order to deal with complex visual question answering, it might be necessary to explicitly incorporate compositional reasoning in the model.

- I.e. Without having seen ”a person touching a bike”, the model should be able to understand the phrase by putting together its understanding of “person”, “bike” and “touching”.

- Different from visual recognition where models learn direct input-output mappings to learn dataset biases
What is visual reasoning?

• **Inputs:**
  An image $x$ and a visual question $q$ about the image

• **Intermediate outputs:**
  A predicted program $z = \pi(q)$ representing the reasoning steps required to answer the question and an execution engine $\phi(x, z)$ executing the program on the image to predict an answer

• **Output:**
  An answer $a \in A$ to the question from a fixed set $A$ of possible answers

Program generator $z$ and execution engine $\phi$
Innovations compared with state-of-arts

- **Module network**: a syntactic parse of a question to determine the architecture of the network
  
  *Existing research*: hand-designed off-the-shelf syntactic parser  
  *Current research*: a learnt program generator that can adapt to the task at hand

- **Semantic parser**
  
  *Existing research*: the semantics of the program and the execution engine are fixed and known a priori  
  *Current research*: learn both the program generator and the execution engine

- **Program-induction methods**
  
  *Existing research*: the interpretation of neural program considers only simple algorithms and program-induction assumes knowledge of the low-level operations  
  *Current research*: the program generator consider inputs comprising an image and an associated question while assume minimal prior knowledge
What is program generator and execution engine?

Programs: focused on learning semantics for a fixed syntax

• Pre-specifying a set $F$ of functions $f$, each of which has a fixed arity $n_f = \{1, 2\}$
• Including in the vocabulary a special constant $Scene$ representing the visual features of the image
• A valid program $z$ is represented as syntax trees where each node contains a function $f$

Execution engine: creating a neural network mapping to each function $f$

• The program $z$ is used to assemble a question-specific neural network composed from a set of modules
• Generic architecture for all unary module, binary module and Scene module
Are there more cubes than yellow things?

- LSTM sequence-to-sequence model
- The resulting sequence of functions is converted to a syntax tree with prefix traversal
- If the sequence is too short, we pad the sequence with Scene constants
- If the sequence is too long, unused functions are discarded
Execution engine

• Scene module takes visual features as input with CNN

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input image</td>
<td>3 × 224 × 224</td>
</tr>
<tr>
<td>Conv(3 × 3, 1024 → 128)</td>
<td>128 × 14 × 14</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Conv(3 × 3, 128 → 128)</td>
<td>128 × 14 × 14</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
</tbody>
</table>

• The final feature map is flattened and passed into a multilayer perception classifier

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final module output</td>
<td>128 × 14 × 14</td>
</tr>
<tr>
<td>Conv(1 × 1, 128 → 512)</td>
<td>512 × 14 × 14</td>
</tr>
<tr>
<td>ReLU</td>
<td>512 × 14 × 14</td>
</tr>
<tr>
<td>MaxPool(2 × 2, stride 2)</td>
<td>512 × 7 × 7</td>
</tr>
<tr>
<td>FullyConnected(512 · 7 · 7 → 1024)</td>
<td>1024</td>
</tr>
<tr>
<td>ReLU</td>
<td>1024</td>
</tr>
<tr>
<td>FullyConnected(1024 →</td>
<td></td>
</tr>
</tbody>
</table>

Are there more cubes than yellow things?
Execution engine

Are there more cubes than yellow things?

- **Unary module**

<table>
<thead>
<tr>
<th>Index</th>
<th>Layer</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Previous module output</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(2)</td>
<td>Conv($3 \times 3$, 128 $\rightarrow$ 128)</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(3)</td>
<td>ReLU</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(4)</td>
<td>Conv($3 \times 3$, 128 $\rightarrow$ 128)</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(5)</td>
<td>Residual: Add (1) and (4)</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(6)</td>
<td>ReLU</td>
<td>$128 \times 14 \times 14$</td>
</tr>
</tbody>
</table>

- **Binary module**

<table>
<thead>
<tr>
<th>Index</th>
<th>Layer</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Previous module output</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(2)</td>
<td>Previous module output</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(3)</td>
<td>Concatenate (1) and (2)</td>
<td>$256 \times 14 \times 14$</td>
</tr>
<tr>
<td>(4)</td>
<td>Conv($1 \times 1$, 256 $\rightarrow$ 128)</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(5)</td>
<td>ReLU</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(6)</td>
<td>Conv($3 \times 3$, 128 $\rightarrow$ 128)</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(7)</td>
<td>ReLU</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(8)</td>
<td>Conv($3 \times 3$, 128 $\rightarrow$ 128)</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(9)</td>
<td>Residual: Add (5) and (8)</td>
<td>$128 \times 14 \times 14$</td>
</tr>
<tr>
<td>(10)</td>
<td>ReLU</td>
<td>$128 \times 14 \times 14$</td>
</tr>
</tbody>
</table>
Figure 3. Visualizations of the norm of the gradient of the sum of the predicted answer scores with respect to the final feature map. From left to right, each question adds a module to the program; the new module is underlined in the question. The visualizations illustrate which objects the model attends to when performing the reasoning steps for question answering. Images are from the validation set.
Training

Separate training with ground-truth programs

• Given VQA dataset containing \((x,q,z,a)\) tuples with ground truth \(z\)
• Use pairs \((q,z)\) of questions and corresponding programs to train the program generator
• Use triplets \((x,z,a)\) of the image, program, and answer to train the execution engine with backpropagation to compute the gradients

Joint training without ground-truth programs

• Use REINFORCE to estimate gradients on the outputs of the program generator.
• The reward for each of its outputs is the negative zero-one loss of the execution engine, with a moving-average baseline.
Training

Semi-supervised learning

Figure 4. Accuracy of predicted programs (left) and answers (right) as we vary the number of ground-truth programs. Blue and green give accuracy before and after joint finetuning; the dashed line shows accuracy of our strongly-supervised model.
### Table 1

Question answering accuracy (higher is better) on the CLEVR dataset for baseline models, humans, and three variants of our model. The strongly supervised variant of our model uses all 700K ground-truth programs for training, whereas the semi-supervised variants use 9K and 18K ground-truth programs, respectively. †Human performance is measured on a 5.5K subset of CLEVR questions.
Experiments

Generalizing to new attribute combinations

**Compositional Generalization Test (CoGenT)**

This data was used in Section 4.7 of the paper to study the ability of models to recognize novel combinations of attributes at test-time. The data is generated in two different conditions:

- **Condition A**
  - Cubes are gray, **blue, brown, or yellow**
  - Cylinders are **red, green, purple, or cyan**
  - Spheres can have any color

- **Condition B**
  - Cubes are **red, green, purple, or cyan**
  - Cylinders are gray, **blue, brown, or yellow**
  - Spheres can have any color
Experiments

Generalizing to new attribute combinations

- Top 1\textsuperscript{st} column:
  Train on A and test on A
- Top 2\textsuperscript{nd} column:
  Train on A and test on B
- Top 3\textsuperscript{rd} column:
  Train A and finetune on B and test on A
- Top 4\textsuperscript{th} column:
  Train A and finetune on B and test on B

- Bottom Figure 1:
  Finetune on B and test on B with overall questions
- Bottom Figure 2:
  Finetune on B and test on B with color-query
- Bottom Figure 3:
  Finetune on B and test on B with shape-query
Experiments

Generalizing to new type of questions

• Able to generalize to questions with **program structures** without observing associated ground-truth programs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Train Short</th>
<th>Finetune Both</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Short</td>
<td>Long</td>
</tr>
<tr>
<td>LSTM</td>
<td>46.4</td>
<td>48.6</td>
</tr>
<tr>
<td>CNN+LSTM</td>
<td>54.0</td>
<td>52.8</td>
</tr>
<tr>
<td>CNN+LSTM+SA+MLP</td>
<td>74.2</td>
<td><strong>64.3</strong></td>
</tr>
<tr>
<td>Ours (25K prog.)</td>
<td><strong>95.9</strong></td>
<td>55.3</td>
</tr>
</tbody>
</table>

Table 2. Question answering accuracy on short and long CLEVR questions. **Left columns:** Models trained only on short questions; our model uses 25K ground-truth short programs. **Right columns:** Models trained on both short and long questions. Our model is trained on short questions then finetuned on the entire dataset; no ground-truth programs are used during finetuning.
Experiments

Human-composed questions

Figure 7. Examples of questions from the CLEVR-Humans dataset, along with predicted programs and answers from our model. Question words that do not appear in CLEVR questions are underlined. Some predicted programs exactly match the semantics of the question (green); some programs closely match the question semantics (yellow), and some programs appear unrelated to the question (red).
Future work

• How to add new modules by automatically identifying and learning without supervision program?
  i.e. “What color is the object with a unique shape?”
  solution: a Turing-complete set of modules

• Control-flow operators could be incorporated into the framework

• Learning programs with limited supervision
Thanks!