Differentiable Programs with Neural Libraries

Gaunt, Brockschmidt, Kushman and Tarlow (2017)

Presented by Ben Striner
Perceptual Programming by Example (PPBE)

- Programming by Example
  - Provide input and output examples
  - Task is to infer a program that satisfies examples
- Perceptual tasks make PPBE
"Illustrative" NTPT Program

Figure 1: Components of an illustrative NTPT program for learning loopy programs that measure path length (path_len) through a maze of street sign images. The learned program (parameterized by instr and goto) must control the position (X, Y) of an agent on a grid of (W×H) street sign images each of size (w×h). The agent has a single register of memory (reg) and learns to interpret street signs using the LOOK neural function. A solution consists of a correctly inferred program and a trained neural network. Learnable components are shown in blue and the NTPT extensions to the TERPRET language are highlighted. The red path on the img_grid shows the desired behavior and is not provided at training time.
Differentiable Programs

• Functions and conditionals differentiable w.r.t. variables

• **Function application.** The statement `z.set_to(foo(x, y))` is translated into
  \[ \mu_i^z = \sum_{jk} I_{ijk} \mu_j^x \mu_k^y \]
  where \( \mu^a \) represents the marginal distribution for the variable \( a \) and \( I \) is an indicator tensor \( \mathbb{1}[i = foo(j,k)] \). This approach extends to all functions mapping any number of integer arguments to an integer output.

• **Conditional statements** The statements `if x == 0: z.set_to(a); elif x == 1: z.set_to(b)` are translated to
  \[ \mu^z = \mu_0^x \mu^a + \mu_1^x \mu^b. \]
  More complex statements follow a similar pattern, with details given in (Gaunt et al., 2016).
Lifelong Learning

- Train model on a sequence of tasks
- Evaluate knowledge transfer
- Evaluate extinction

Figure 2: Overview of tasks in the (a) ADD2x2, (b) APPLY2x2 and (c) MATH scenarios. ‘A’ denotes the APPLY operator which replaces the ? tiles with the selected operators and executes the sum. We show two MATH examples of different length.
Add2x2

• Given MNIST digits and operator indicators, calculate output

```python
# initialization:
R0 = READ

# program:
R1 = MOVE_EAST
R2 = MOVE_SOUTH
R3 = SUM(R0, R1)
R4 = NOOP
return R3
```

- **NOOP**: a trivial no-operation instruction.
- **MOVE_NORTH, MOVE_EAST, MOVE_SOUTH, MOVE_WEST**: translate the head (if possible) and return the result of applying the neural network chosen by `net_choice` to the image in the new cell.
- **ADD (·,·)**: accepts two register addresses and returns the sum of their contents.
Apply2x2

- Given handwritten operators and digit indicators, calculate output
- APPLY instead of ADD
Math!

• Given mnist digits and handwritten operators of variable lengths, calculate an output

(c)

\[
\begin{array}{c}
6 / 3 = 2 \\
7 \times 1 + 4 = 11
\end{array}
\]
Knowledge Transfer

• Two NN functions shared between scenarios
  • Both 2 layer relu with softmax
  • One with 10 outputs, one with 4

• Operator and MNIST recognition can be shared

• Only one new net can be trained at a time
Baseline Models

• Baseline for task 1 and 2 is a simple MLP
• Operates on concatenated features
• Baseline task 3 is LSTM

- Each of the images in the $2 \times 2$ grid is passed through an embedding network with 2 layers of 256 neurons (cf. net_0/1) to produce a 10-dimensional embedding. The weights of the embedding network are shared across all 4 images.

- These 4 embeddings are concatenated into a 40-dimensional vector and for the APPLY2x2 the auxiliary integers are represented as one-hot vectors and concatenated with this 40-dimensional vector.

- This is then passed through a network consisting of 3 hidden layers of 128 neurons to produce a 19-dimensional output.
Baseline Models (Cartoon Version)

Figure 5: Cartoon illustration of all models used in the $2 \times 2$ experiments. See text for details.
## Results

<table>
<thead>
<tr>
<th></th>
<th>task</th>
<th>indep</th>
<th>PNN</th>
<th>MTNN-1</th>
<th>MTNN-2</th>
<th>NTPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD2x2</td>
<td>top</td>
<td>35%</td>
<td>35%</td>
<td>26%</td>
<td>24%</td>
<td>87%</td>
</tr>
<tr>
<td></td>
<td>left</td>
<td>32%</td>
<td>36%</td>
<td>38%</td>
<td>47%</td>
<td>87%</td>
</tr>
<tr>
<td></td>
<td>bottom</td>
<td>34%</td>
<td>33%</td>
<td>40%</td>
<td>56%</td>
<td>86%</td>
</tr>
<tr>
<td></td>
<td>right</td>
<td>32%</td>
<td>35%</td>
<td>44%</td>
<td>60%</td>
<td>86%</td>
</tr>
<tr>
<td>APPLY2x2</td>
<td>top</td>
<td>38%</td>
<td>39%</td>
<td>40%</td>
<td>38%</td>
<td>98%</td>
</tr>
<tr>
<td></td>
<td>left</td>
<td>39%</td>
<td>51%</td>
<td>41%</td>
<td>39%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>bottom</td>
<td>39%</td>
<td>48%</td>
<td>41%</td>
<td>40%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>right</td>
<td>39%</td>
<td>51%</td>
<td>42%</td>
<td>37%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 7: Final accuracies on all $2 \times 2$ tasks for all models at the end of lifelong learning
Figure 8: Generalization behavior on MATH expressions. Solid dots indicate expression lengths used in training. We show results on (a) a simpler non-perceptual MATH task (numbers in parentheses indicate parameter count in each model) and (b) the MATH task including perception.
Optimization Difficulties

• Train on expressions with 2 digits
• 2/100 random restarts converge
• Loopy program “provably generalizes perfectly” to longer sequences
Avoiding Forgetting

• Set learning rate of perceptual parts to 1/100 of task-specific parts
Details

• Probably minimizing cross entropy with correct answer
• Trains using expectation over instructions
• Each variable is a distribution over integers
Thank you

Questions/Discussion?