# Differentiable Programs with Neural Libraries

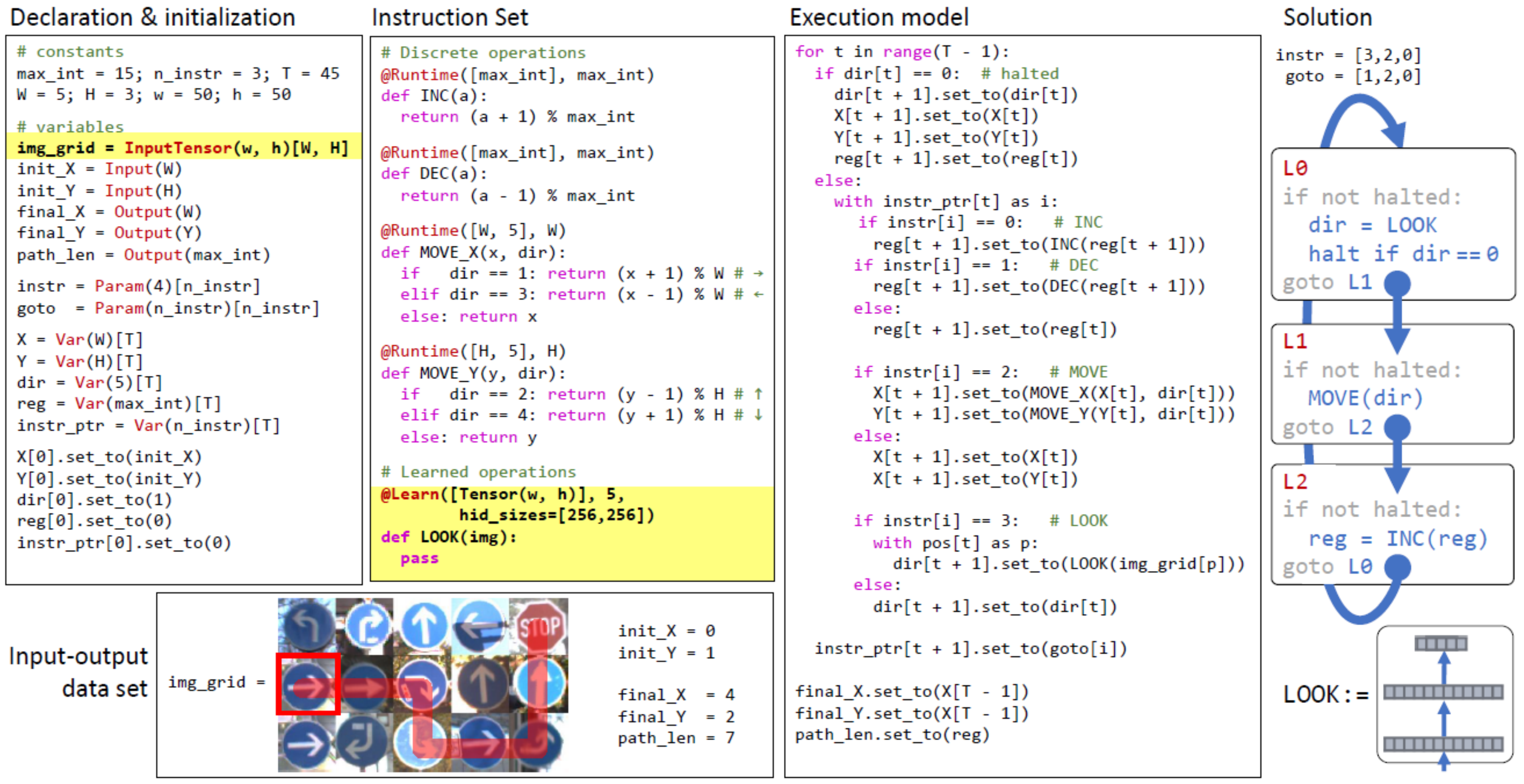
Gaunt, Brockschmidt, Kushman and Tarlow (2017)

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



Declaration & initialization

```
# constants
max_int = 15; n_instr = 3; T = 45
W = 5; H = 3; w = 50; h = 50

# variables
img_grid = InputTensor(w, h)[W, H]
init_X = Input(W)
init_Y = Input(H)
final_X = Output(W)
final_Y = Output(Y)
path_len = Output(max_int)

instr = Param(4)[n_instr]
goto  = Param(n_instr)[n_instr]

X = Var(W)[T]
Y = Var(H)[T]
dir = Var(5)[T]
reg = Var(max_int)[T]
instr_ptr = Var(n_instr)[T]

X[0].set_to(init_X)
Y[0].set_to(init_Y)
dir[0].set_to(1)
reg[0].set_to(0)
instr_ptr[0].set_to(0)
```

Instruction Set

```
# Discrete operations
@Runtime([max_int], max_int)
def INC(a):
  return (a + 1) % max_int

@Runtime([max_int], max_int)
def DEC(a):
  return (a - 1) % max_int

@Runtime([W, 5], W)
def MOVE_X(x, dir):
  if   dir == 1: return (x + 1) % W # →
  elif dir == 3: return (x - 1) % W # ←
  else: return x

@Runtime([H, 5], H)
def MOVE_Y(y, dir):
  if   dir == 2: return (y - 1) % H # ↑
  elif dir == 4: return (y + 1) % H # ↓
  else: return y

# Learned operations
@Learn([Tensor(w, h)], 5,
          hid_sizes=[256,256])
def LOOK(img):
  pass
```

Execution model

```
for t in range(T - 1):
  if dir[t] == 0:  # halted
    dir[t + 1].set_to(dir[t])
    X[t + 1].set_to(X[t])
    Y[t + 1].set_to(Y[t])
    reg[t + 1].set_to(reg[t])
  else:
    with instr_ptr[t] as i:
      if instr[i] == 0:   # INC
        reg[t + 1].set_to(INC(reg[t + 1]))
      if instr[i] == 1:   # DEC
        reg[t + 1].set_to(DEC(reg[t + 1]))
      else:
        reg[t + 1].set_to(reg[t])

      if instr[i] == 2:   # MOVE
        X[t + 1].set_to(MOVE_X(X[t], dir[t]))
        Y[t + 1].set_to(MOVE_Y(Y[t], dir[t]))
      else:
        X[t + 1].set_to(X[t])
        X[t + 1].set_to(Y[t])

      if instr[i] == 3:   # LOOK
        with pos[t] as p:
          dir[t + 1].set_to(LOOK(img_grid[p]))
      else:
        dir[t + 1].set_to(dir[t])

  instr_ptr[t + 1].set_to(goto[i])

final_X.set_to(X[T - 1])
final_Y.set_to(X[T - 1])
path_len.set_to(reg)
```

Solution

```
instr = [3,2,0]
 goto = [1,2,0]

L0
if not halted:
  dir = LOOK
  halt if dir == 0
goto L1

L1
if not halted:
  MOVE(dir)
goto L2

L2
if not halted:
  reg = INC(reg)
goto L0
```

LOOK :=

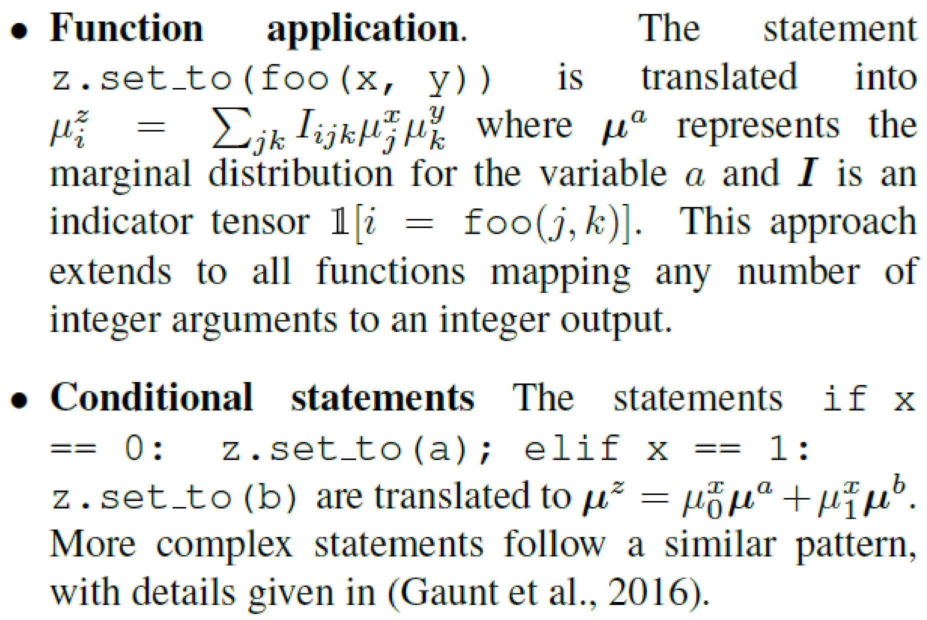
Input-output data set

```
img_grid =

init_X = 0
init_Y = 1

final_X  = 4
final_Y  = 2
path_len = 7
```

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 $\boldsymbol{\mu}^a$ represents the marginal distribution for the variable $a$ and $\boldsymbol{I}$ is an indicator tensor $\mathbb{1}[i = \texttt{foo}(j,k)]$. 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 $\boldsymbol{\mu}^z = \mu_0^x \boldsymbol{\mu}^a + \mu_1^x \boldsymbol{\mu}^b$. 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

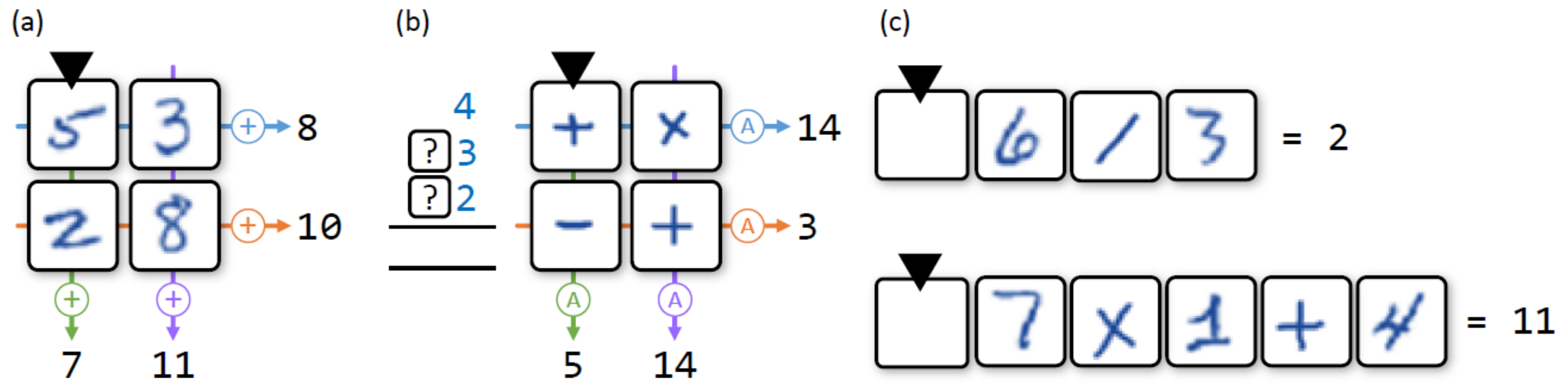
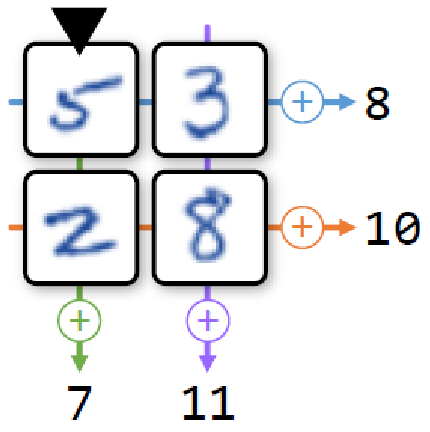


Figure 2: Overview of tasks in the (a) ADD2X2, (b) APPLY2X2 and (c) MATH scenarios. 'A' denotes the APPLY operator which replaces the ? tiles with the selected operators and executes the sum. We show two MATH examples of different length.

# Add2x2

- Given MNIST digits and operator indicators, calculate output

(a)



(a)

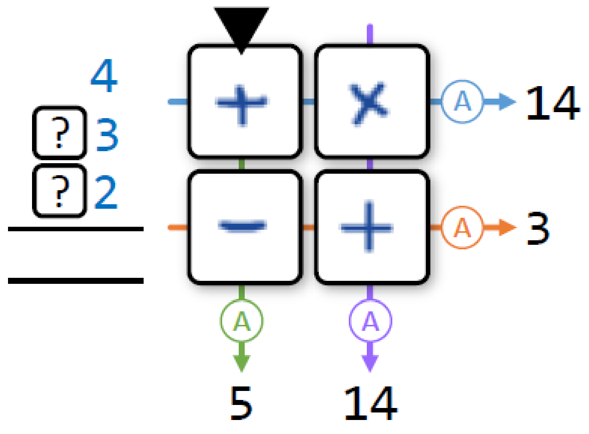
```
# initialization:
R0 = READ
# program:
R1 = MOVE_EAST
R2 = MOVE_SOUTH
R3 = SUM(R0, R1)
R4 = NOOP
return R3
```

- 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

(b)

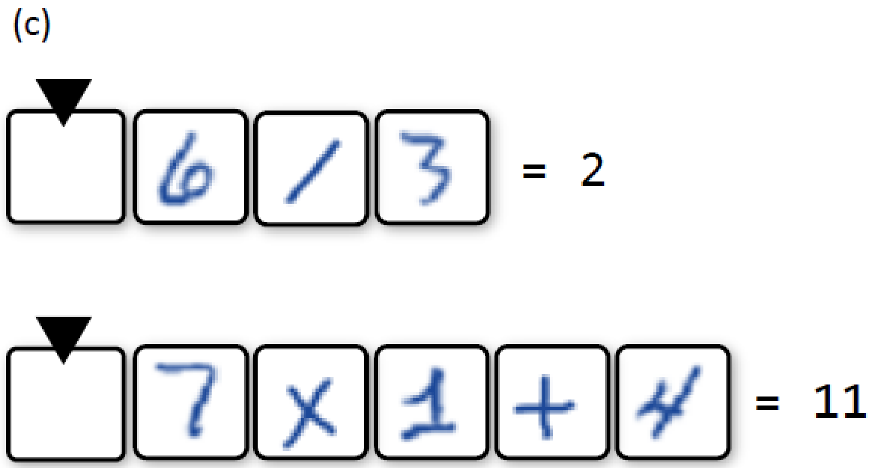


(b)

```
# initialization:
R0 = InputInt[0]
R1 = InputInt[1]
R2 = InputInt[2]
R3 = READ
# program:
R4 = MOVE_EAST
R5 = MOVE_SOUTH
R6 = APPLY(R0, R1, R4)
R7 = APPLY(R6, R2, R5)
return R7
```

# Math!

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

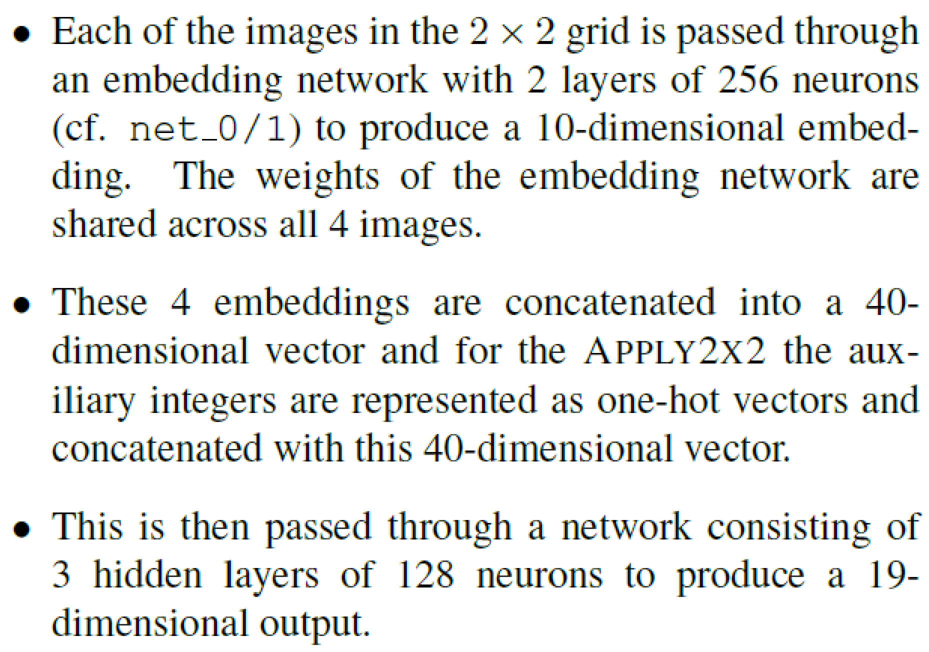


## Knowledge Transfer

- 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)

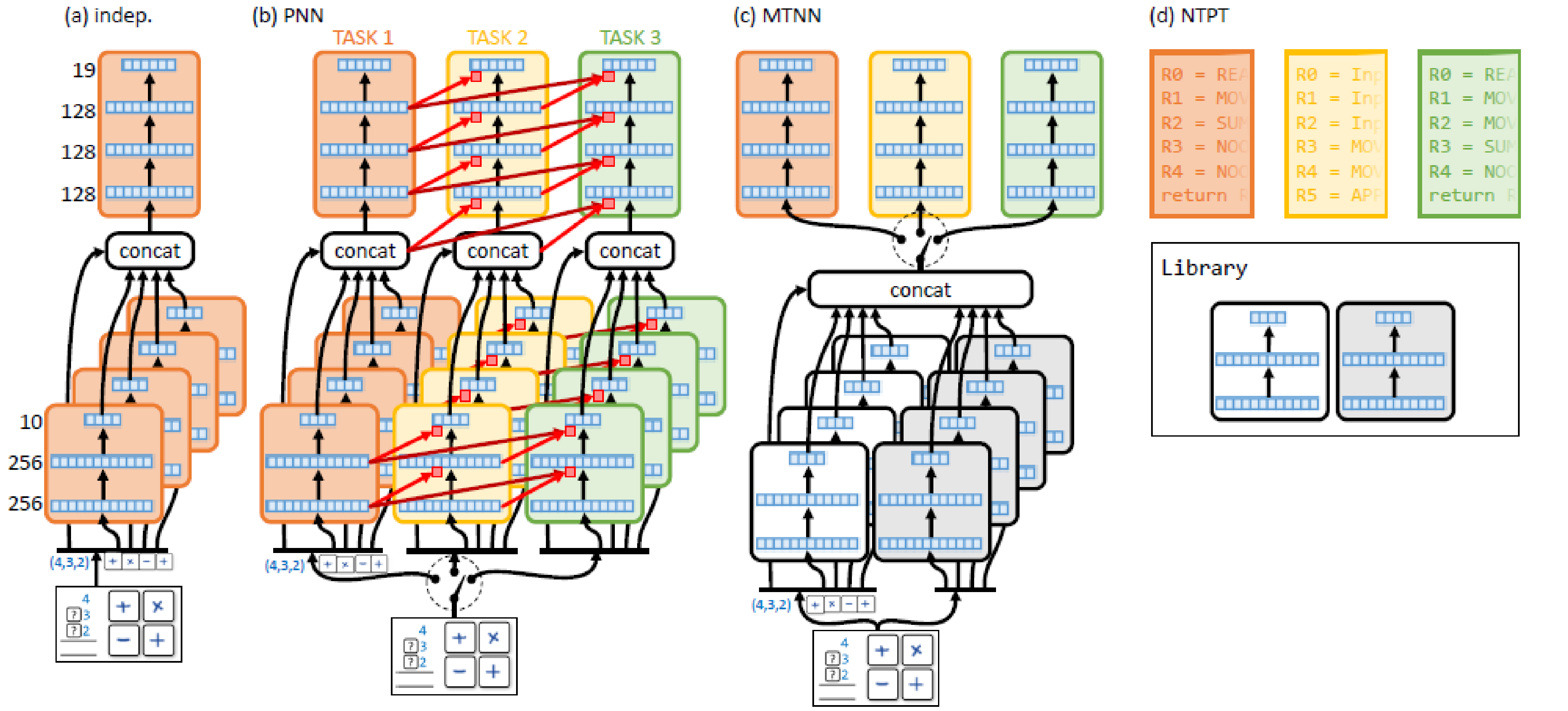
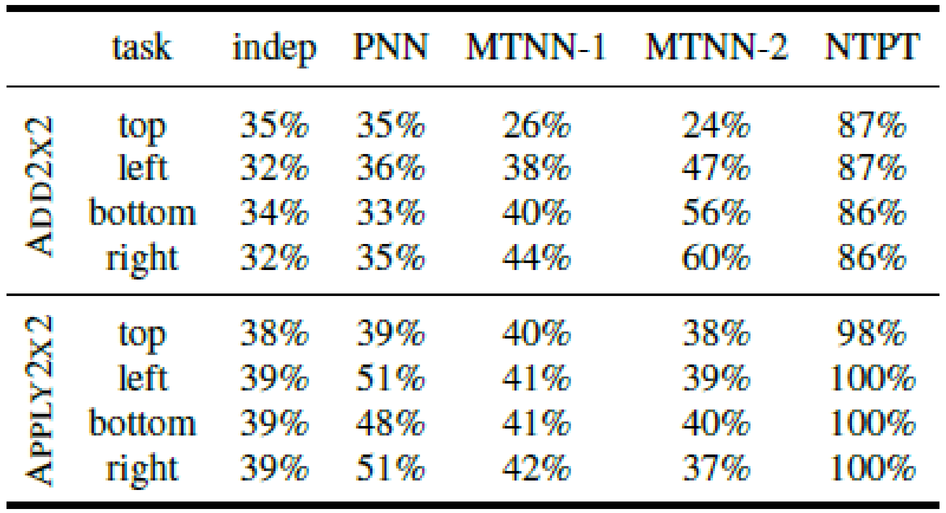


Figure 5: Cartoon illustration of all models used in the 2 × 2 experiments. See text for details.

## Results



| | task | indep | PNN | MTNN-1 | MTNN-2 | NTPT |
|---|---|---|---|---|---|---|
| ADD2X2 | top | 35% | 35% | 26% | 24% | 87% |
| | left | 32% | 36% | 38% | 47% | 87% |
| | bottom | 34% | 33% | 40% | 56% | 86% |
| | right | 32% | 35% | 44% | 60% | 86% |
| APPLY2X2 | top | 38% | 39% | 40% | 38% | 98% |
| | left | 39% | 51% | 41% | 39% | 100% |
| | bottom | 39% | 48% | 41% | 40% | 100% |
| | right | 39% | 51% | 42% | 37% | 100% |

Figure 7: Final accuracies on all $2 \times 2$ tasks for all models at the end of lifelong learning

# Generalization

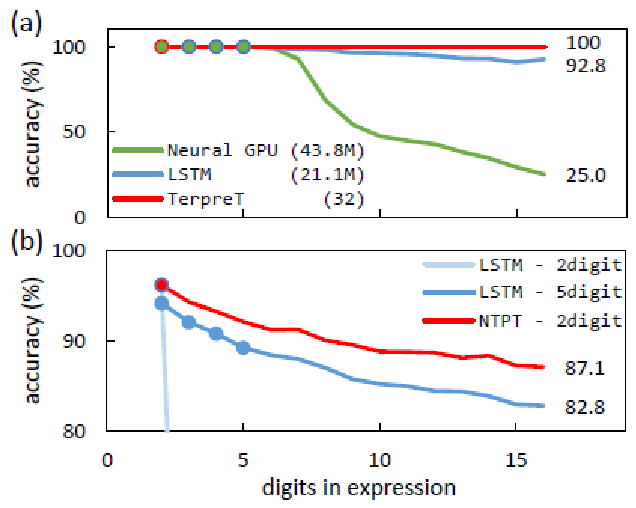


Figure 8: Generalization behavior on MATH expressions. Solid dots indicate expression lengths used in training. We show results on (a) a simpler non-perceptual MATH task (numbers in parentheses indicate parameter count in each model) and (b) the MATH task including perception.

## 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?