**Research Log:**

**1)I define the model and the LSTM in the article:**

@knet function wbf3(x1, x2, x3; f=:sigm, o...)

y1 = wdot(x1; o...)

y2 = wdot(x2; o...)

y3 = wdot(x3, o...)

x3 = add(y2,y1)

x4 = add(x3,y3)

y4 = bias(x4; o...)

return f(y4; o...)

end

@knet function my\_lstm(x; fbias=0.08, o...)

input = wbf3(x,h,cell; o..., f=:sigm, binit=Uniform(-fbias,fbias))

forget = wbf3(x,h,cell; o..., f=:sigm, binit=Uniform(-fbias,fbias))

newmem = wbf2(x,h; o..., f=:tanh, binit=Constant(-fbias,fbias))

cell = input .\* newmem + cell .\* forget

output = wbf3(x,h,cell; o..., f=:sigm, binit=Constant(-fbias,fbias))

h = tanh(cell) .\* output

return h

end

@knet function my\_model(x; fbias=0, o...)

h1 = lstm(x: out=400)

h2 = lstm(h1: out=400)

return h2

end

**2)I write my input generator code in python(incomplete):**

import sys

import random

args = sys.argv

class Operation:

   def \_\_init\_\_(self):

       self.operation = ''

       self.params = []

   def evaluate(self):

       op = self.operation

       if op == '+':

           return params[0] + params[1]

       elif op == '-':

           return params[0]-params[1]

       elif op == '\*':

           return params[0]\*params[1]

       elif op == '<':

           return params[0]<params[1]

       elif op == '>':

           return params[0]>params[1]

       elif op == '==':

           return params[0]==params[1]

       elif op == 'for-':

           my\_value=params[1]

           for x in range(params[0]):my\_value-=params[2]

           return my\_value

        elif op == 'for+':

           my\_value=params[1]

           for x in range(params[0]):my\_value+=params[2]

           return my\_value

   @property

   def getparamcount(self):

       op = self.operation

       if op == '+':

           return [0,0]

       elif op == '-':

           return 2

       elif op == '\*':

           return 2

       elif op == '<':

           return 2

           return 2

       elif op == '>':

           return 2

       elif op == '==':

           return 2

       elif op == 'for-':

           return 3

       elif op == 'for+':

           return 3

length = int(args[1])

nesting = int(args[2])

Input = length, nesting

stack = []

Operations = ['+', '-', '\*', '<','>','==', 'for-', '=', 'for+']

for i in range (1,nesting):

   Operation = random.choice(Operation)

   Values = []

   Code = []

   for params in Operation.param

       if stack.count()!=0 and random.uniform(0,1) > 0.5:

           value, code = stack.pop()

       else:

           value = random.int(10^length,10^(length+1)-1)

           code = str(value)

       Values.append(value)

       Code.append(code)

   new\_value= Operation.evaluate(Values)

   new\_code = Operation.generate code(codes) stack.push((new value, new code))

final value, final code = stack.pop()

datasets = training, validation, testing

idx = hash(final code) modulo 3

datasets[idx].add((final value, final code))

3) I opened an issue about backpropagation issue in Author’s git repository.

https://github.com/wojciechz/learning\_to\_execute/issues/12