# Lecture 9: Inductive Program Synthesis via Deep Learning

Yu Feng

Spring 2021

# Summary of previous lecture

- R3 was out
- HW2 was out
- Proposal is out
- Synthesis with abstract semantics

# Outline for today

- Program synthesis with DNN
- What’s next? Multi-model synthesis from Yanju Chen on Wednesday

For all inputs x, find a program P that meets the specification $\phi$

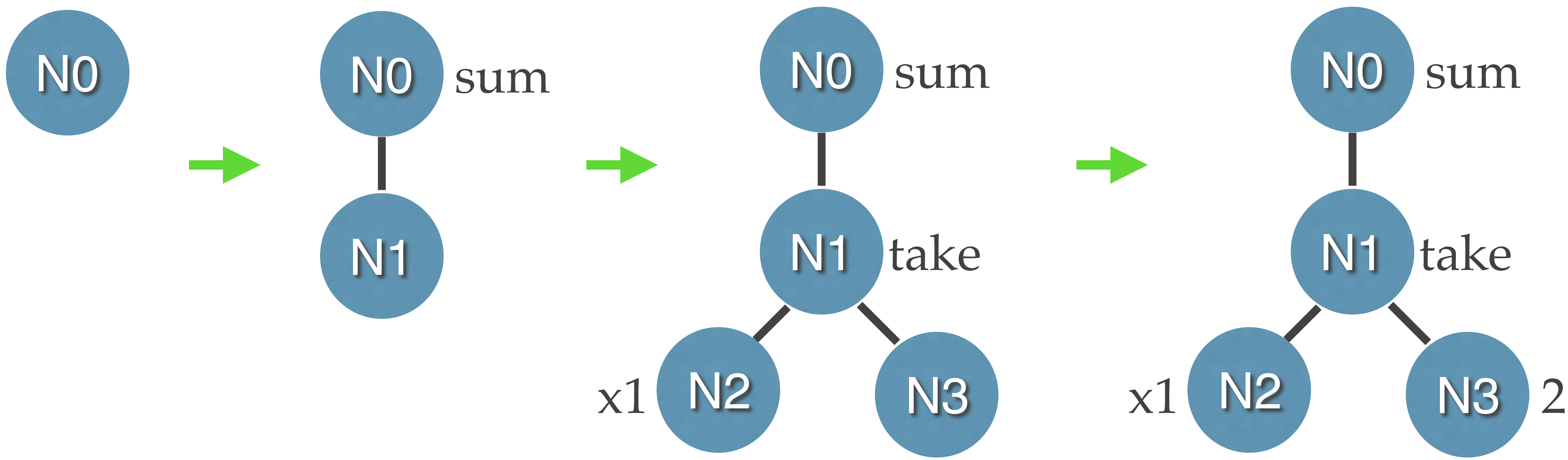
$$\exists\, P.\ \forall\, x.\ \phi(x, P(x))$$

$$
\begin{aligned}
N \rightarrow\ & \texttt{0 | ... | 10 |}\ x_i \mid \texttt{last}(L) \mid \texttt{head}(L) \mid \texttt{sum}(L) \\
& \mid \texttt{maximum}(L) \mid \texttt{minimum}(L) \\
L \rightarrow\ & \texttt{take}(L, N) \mid \texttt{filter}(L, T) \mid \texttt{sort}(L) \mid \texttt{reverse}(L) \mid x_i \\
T \rightarrow\ & \texttt{geqz | leqz | eqz}
\end{aligned}
$$

$$N \rightarrow \texttt{0} \mid \texttt{...} \mid \texttt{10} \mid x_i \mid \texttt{last}(L) \mid \texttt{head}(L) \mid \texttt{sum}(L) \mid \texttt{maximum}(L) \mid \texttt{minimum}(L)$$

$$L \rightarrow \texttt{take}(L, N) \mid \texttt{filter}(L, T) \mid \texttt{sort}(L) \mid \texttt{reverse}(L) \mid x_i$$

$$T \rightarrow \texttt{geqz} \mid \texttt{leqz} \mid \texttt{eqz}$$



# Top-down enumerative synthesis

```
Synthesize(inputs, outputs){
 wlist := start_symbol
 while(true):
  Deque p from wlist;
  if(isConcrete(p))
   if(isCorrect(p, inputs, outputs))
    return p;
  else
   wlist := wlist U grow(p);
}
```

Which one we should pick?

How to grow?

“A dream of artificial intelligence is to build systems that can write computer programs”

[1] DEEPCODER: LEARNING TO WRITE PROGRAMS. Balog et al., ICLR’17

# DSL for list manipulation

$$N \to \texttt{0} \mid \texttt{...} \mid \texttt{10} \mid x_i \mid \texttt{last}(L) \mid \texttt{head}(L) \mid \texttt{sum}(L) \mid \texttt{maximum}(L) \mid \texttt{minimum}(L)$$

$$L \to \texttt{take}(L, N) \mid \texttt{filter}(L, T) \mid \texttt{sort}(L) \mid \texttt{reverse}(L) \mid x_i$$

$$T \to \texttt{geqz} \mid \texttt{leqz} \mid \texttt{eqz}$$

```
a ← [int]
b ← SORT a
c ← FILTER (>0) b
d ← HEAD c
e ← DROP d b
```

**An input-output example:**

*Input*:

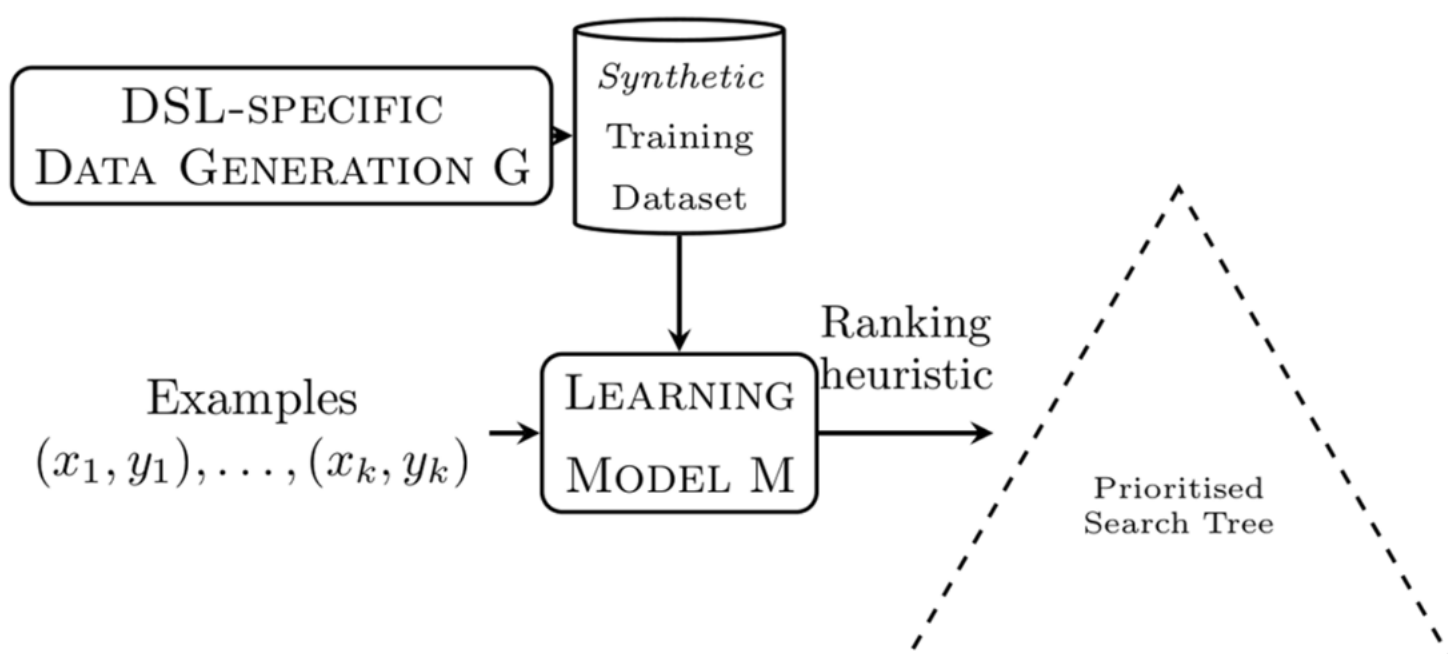
[-17, -3, 3, 11, 0, -5, -9, 13, 6, 6, -8, 11]

*Output*:

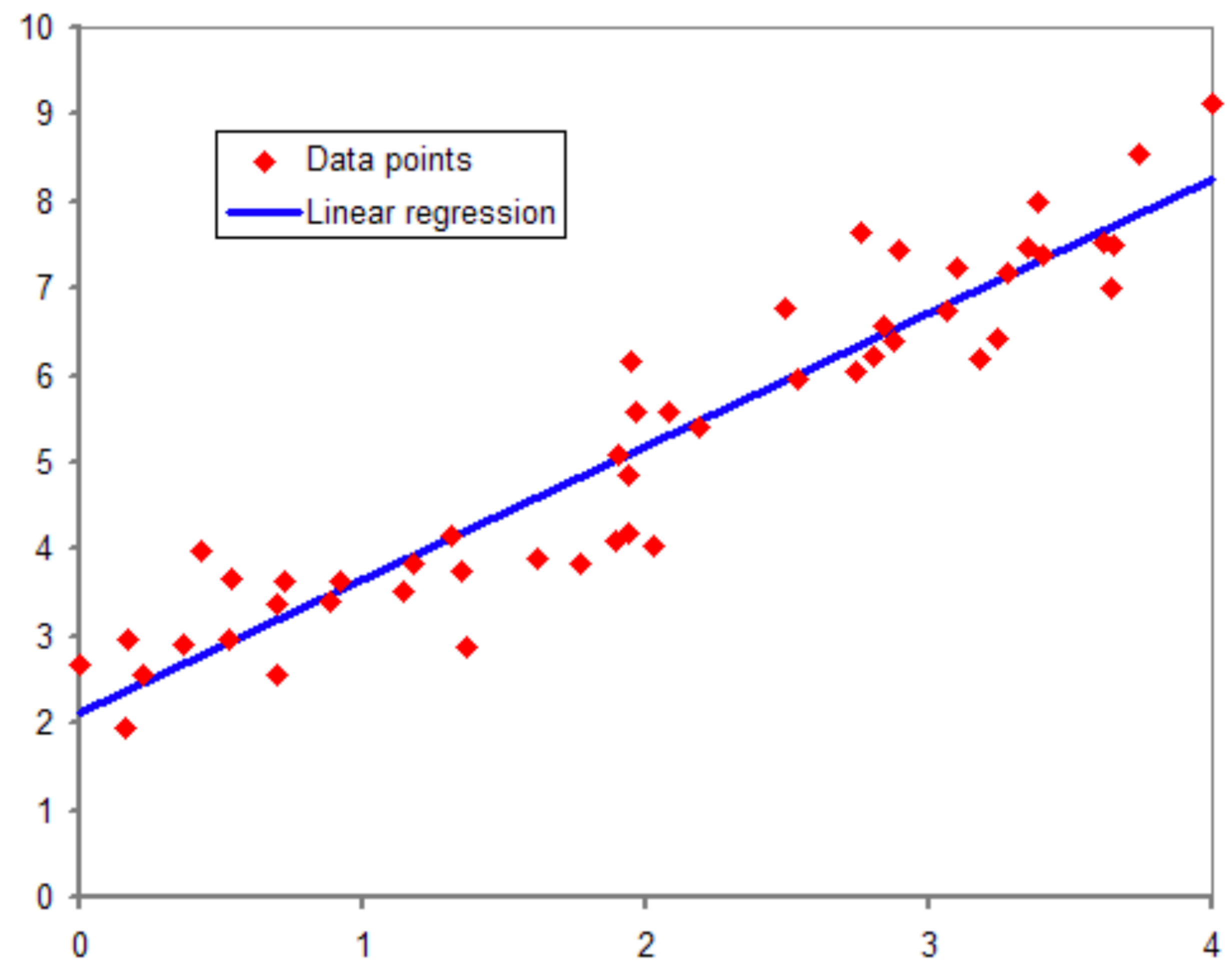
[-5, -3, 0, 3, 6, 6, 11, 11, 13]

# ML for synthesis

- Not all programs are equally likely
- The I/O give some indications about the program



# Linear regression

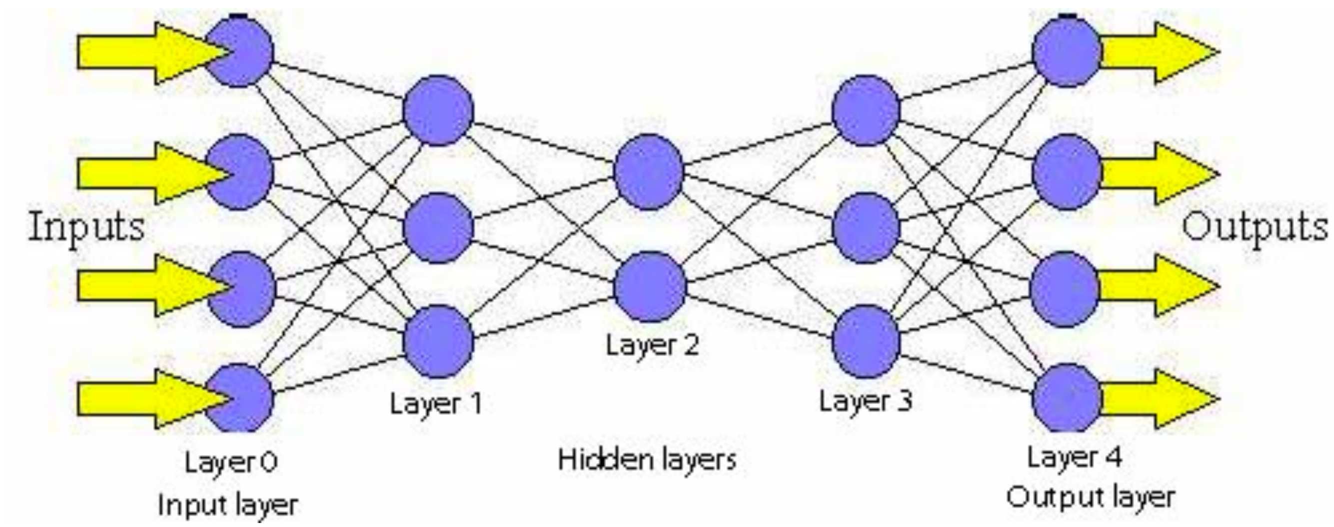



**y=ax+b**

https://en.wikipedia.org/wiki/Regression_analysis

# Feedforward neural network

I/O examples




Likelihood of each function

https://en.wikipedia.org/wiki/Feedforward_neural_network

# ML for synthesis

- The neural network takes as input the I/O and outputs for each function the likelihood (a number in [0,1]) that the function is used in a program satisfying these I/O. The search procedure is then a biassed DFS: the most likely functions according to the neural network are tried first

| (+1) | (-1) | (*2) | (/2) | (*-1) | (**2) | (*3) | (/3) | (*4) | (/4) | (>0) | (>0) | (%2==1) | (%2==0) | HEAD | LAST | MAP | FILTER | SORT | REVERSE | TAKE | DROP | ACCESS | ZIPWITH | SCANL1 | + | - | * | MIN | MAX | COUNT | MINIMUM | MAXIMUM | SUM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| .0 | .0 | .1 | .0 | .0 | .0 | .0 | .0 | 1.0 | .0 | .0 | 1.0 | .0 | .2 | .0 | .0 | 1.0 | 1.0 | 1.0 | .7 | .0 | .1 | .0 | .4 | .0 | .0 | .1 | .0 | .2 | .1 | .0 | .0 | .0 | .0 |

Figure 2: Neural network predicts the probability of each function appearing in the source code.

# Data generation

- An important question is: how do we train the neural network
- The obvious answer is: using millions of programs and I/O.
- But such a dataset may not be available. Hence the actual answer is to generate millions of programs and I/O
- Arguably the most challenging part

# Evaluation

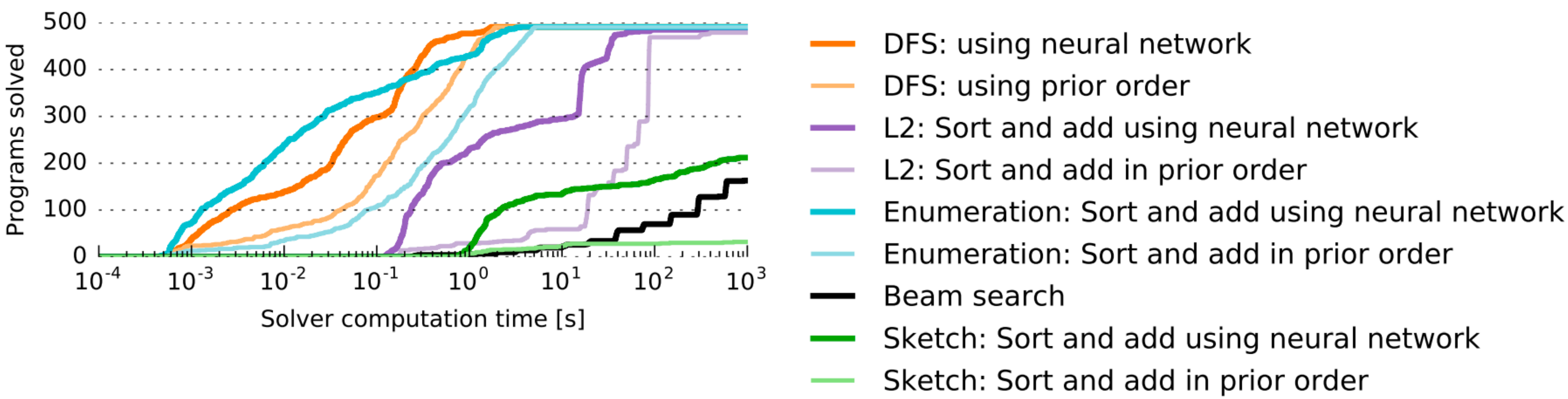



Figure 5: Number of test problems solved versus computation time.

# TODOs by next lecture

- Start to work on your final project and proposal