Improvements on Description-based Neural Program Synthesis Models

Walden Yan

Mentor: William Moses

Benefits of an AI that can Program

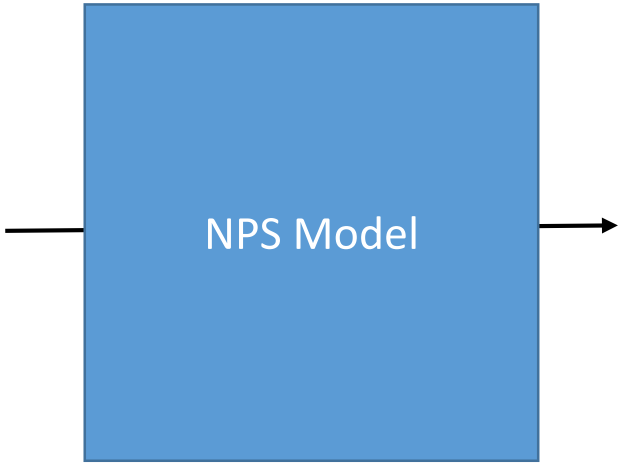
- Accelerate software development
- Quickly verify theoretical work
- Arbitrary capability voice assistants
- Much More

Description-Based Neural Program Synthesis

- Use an artificial neural network to generate or aid in the automatic generation of a program given some text description of what it should do

Ex.

"You are given an array *a*. Find the smallest element in *a* which is strictly greater than the minimum element in *a*".



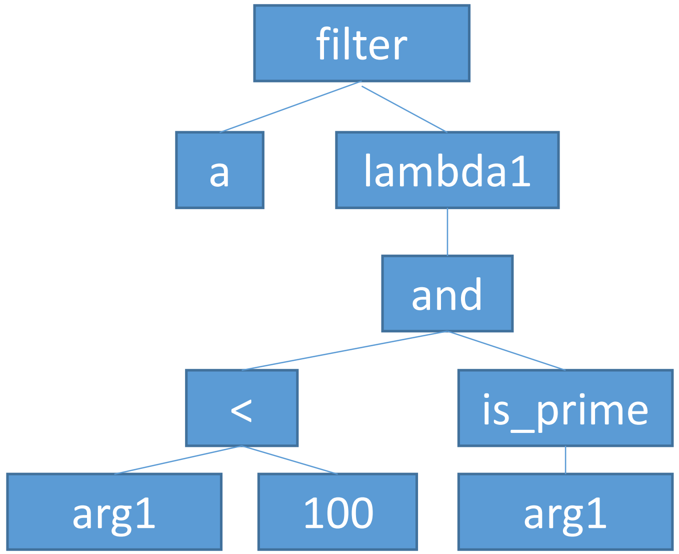

```
(reduce (filter a
(partial0 (reduce a inf min) <))
inf min)
```

AlgoLisp Dataset

- Dataset of 100 thousand English-text problem statements and model solutions
- Input-output pairs to test against
  - 10% of programs don’t pass their I/O pairs so it is common to use a cleaned version of the dataset
- Complexity and Large Search Space
  - Impractical to derive programs from test data

AlgoLisp Dataset

- Solutions are written in a Lisp-inspired DSL
  - Prefix notation
  - Programs have a natural tree structure



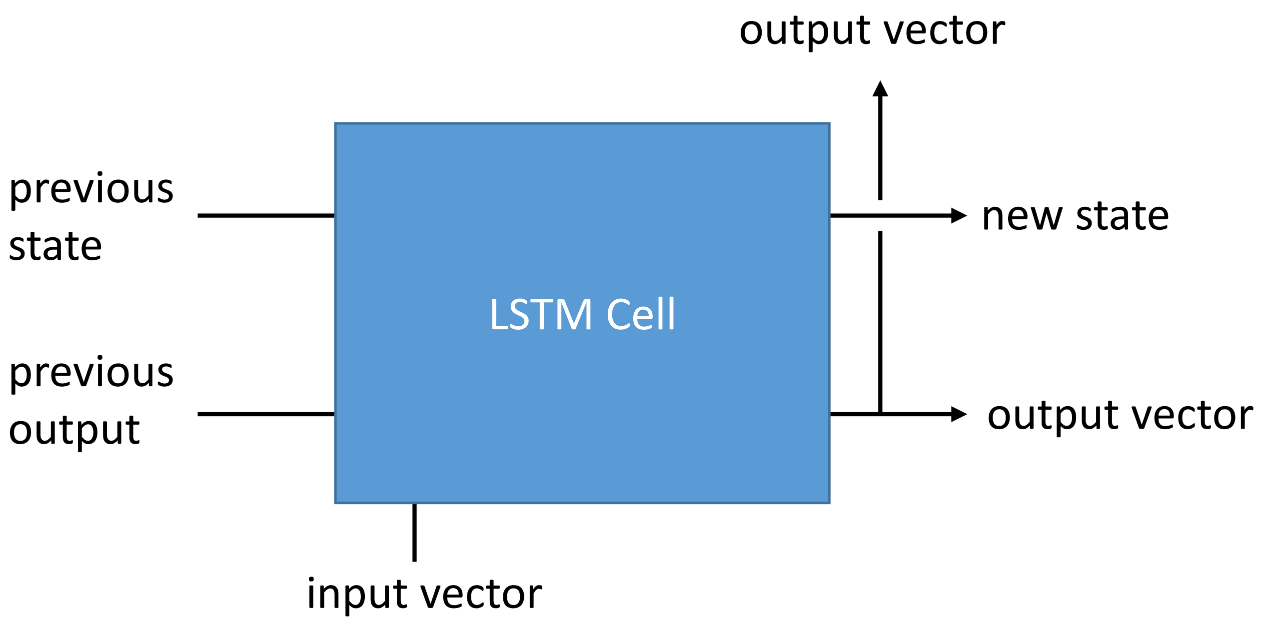

```
filter a lambda1 and < arg1 100 is_prime arg1
```

```
(filter a (lambda1 (and (< arg1 100) (is_prime arg1))))
```

Basic Model

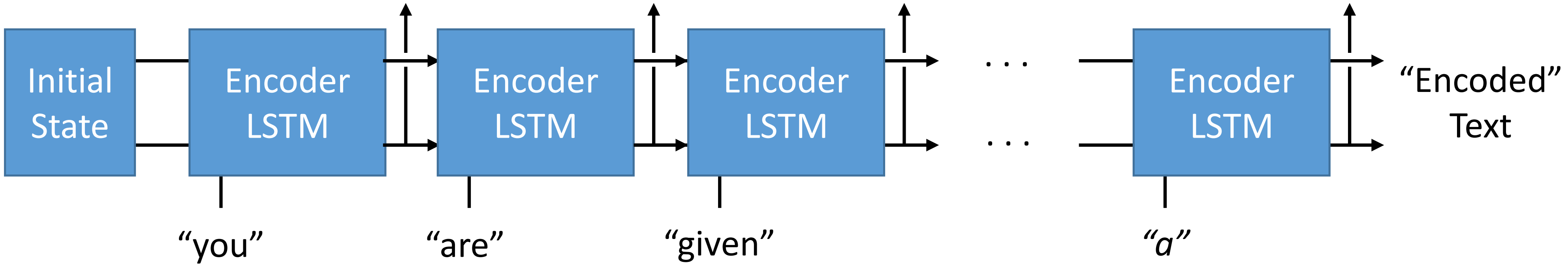
Simple LSTM Model (Seq2Seq)

- Goal: Try to predict just the next token in the program



Simple LSTM Model (Seq2Seq)

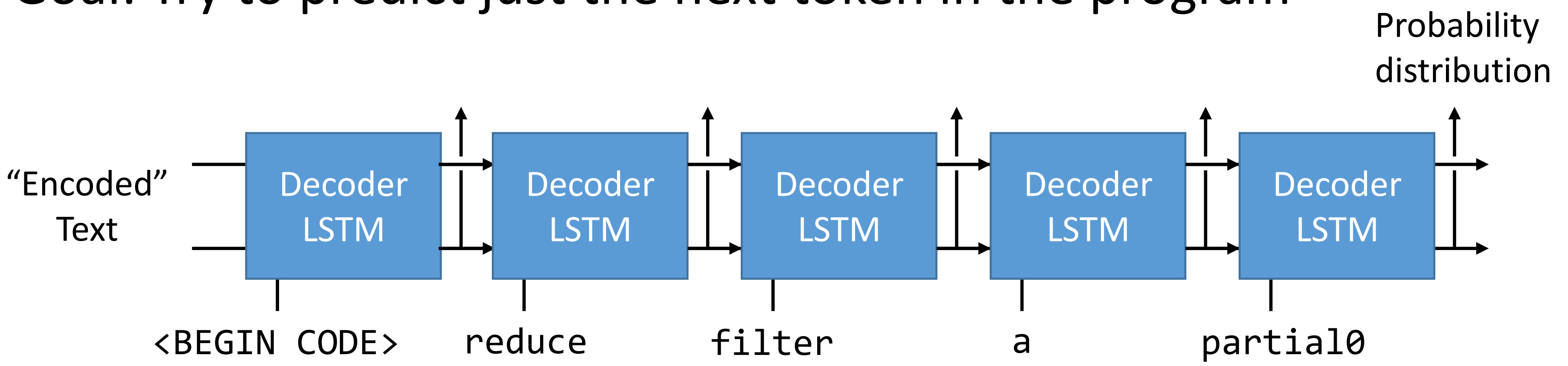
- Goal: Try to predict just the next token in the program



“You are given an array *a*.Find the smallest element in *a* which is strictly greater than the minimum element in *a*”.

Simple LSTM Model (Seq2Seq)

- Goal: Try to predict just the next token in the program




```
(reduce (filter a (partial0 ? . . .
```

Simple LSTM Model (Seq2Seq)

- Goal: Try to predict just the next token in the program

```
                                         97%   (reduce (filter a (partial0 (reduce

                                         0.5%  (reduce (filter a (partial0 (lambda2
(reduce (filter a (partial0
                                         0.3%  (reduce (filter a (partial0 (filter

                                          .
                                          .
                                          .
```

Simple LSTM Model (Seq2Seq)

- Repeat

`(reduce (filter a (partial0`

97% `(reduce (filter a (partial0 (reduce`

0.5% `(reduce (filter a (partial0 (lambda2`

0.3% `(reduce (filter a (partial0 (filter`

.
.
.

Simple LSTM Model (Seq2Seq)

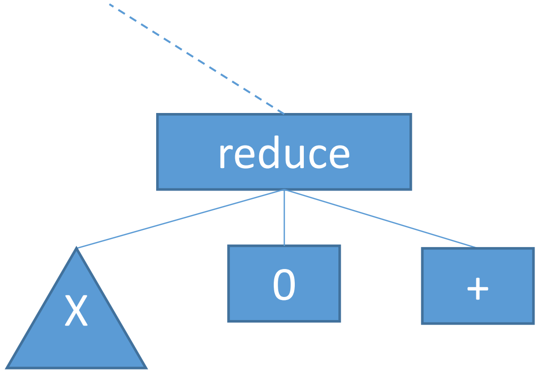
- Try to find top K (beam size) most likely candidates
  - Greedily keep track of K most likely candidates at each level

```
(reduce (filter a (partial0

97%   (reduce (filter a (partial0 (reduce
0.5%  (reduce (filter a (partial0 (lambda2
0.3%  (reduce (filter a (partial0 (filter
.
.
.
```

Simple LSTM Model (Seq2Seq)

- But these do not work that well
  - It's hard to effectively encode the text in one sweep
  - Syntax rules are hard for neural networks to learn
  - Forces programs into a linear structure




```
...(reduce (code for calculating x) 0 +)...
```

Attempt to resolve via more complex recurrence structure
Ex. Seq2Tree (Polosukhin, Skidanov 2018)

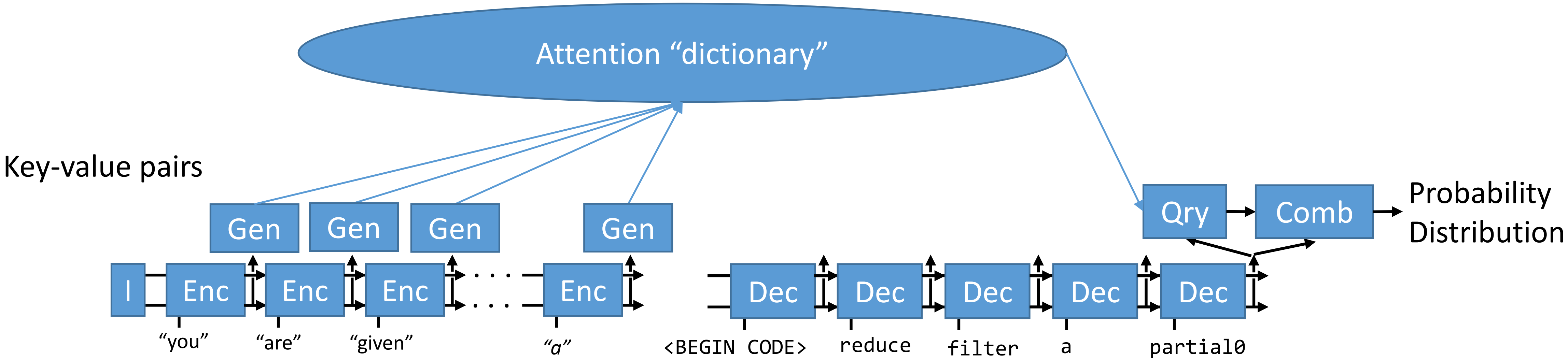
Our Modifications

Our Approach

1. Attention Mechanisms	forgetting information during encoding
2. Learned Syntax Layer	syntactically invalid programs
3. Token Pairing (novel)	linear structure

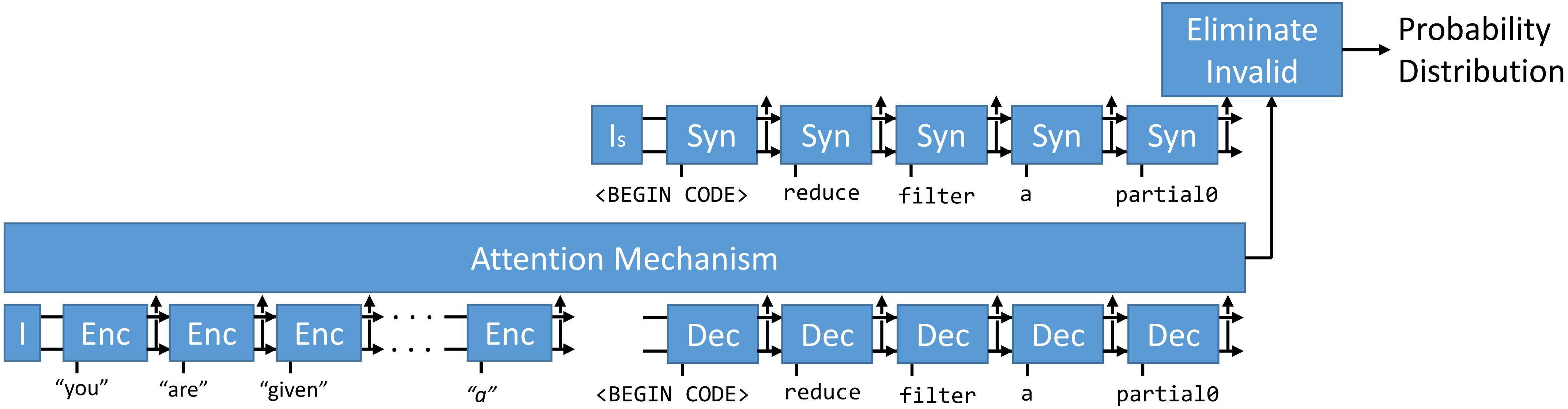
Attention Mechanisms

- Mechanism that allows decoder to focus in on specific sections of the text



Learned Syntax Layer [1]

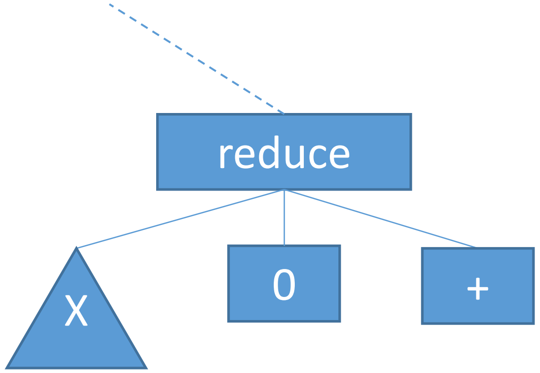
- Jointly trained LSTM that is motivated to recognize syntactically invalid options

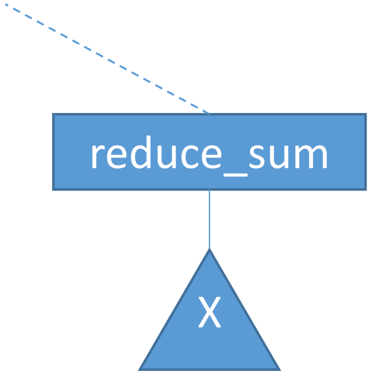


[1] Bunel, Hausknecht, Devlin, Singh, Kohli 2018

Token Pairing

- Similar to the practice of Byte Pair Encoding common in NLP
- Create new tokens to represent common patterns in the code trees






```
...(reduce (code for calculating x) 0 +)...   ...(reduce_sum (code for calculating x))...
```

Token Pairing

- Repeatedly combine most common pair of adjacent tokens

```
# replace tokens with integers 0 ... (number of unique tokens – 1)
training_programs = encode(TRAINING_PROGRAMS)

for _ in range(NUM_ITERATIONS):
    # dict of (pair, int)
    freq_dict = create_adjacency_freq_dict(training_programs)

    most_freq_pair = max(freq_dict, key=freq_dict.get)
    training_programs = [
        replace_pair(program, most_freq_pair, num_unique_tokens)
        for program in training_programs
    ]
    num_unique_tokens += 1
```

Simplified for your viewing pleasure

Token Pairing

- Sub-procedures that consist of tokens not adjacent in the in-order traversal are easier for the model to recreate

- Programs can become shorter
  - Training is more stable in earlier stages
    - Can be circumvented with curriculum training
  - Less depth to the beam search
    - But also more branching at each level

Experiments

Performance

Model	Dev Accuracy	Test Accuracy	Parameters
Model_81 (No Token Pairing)	**97.7%**	**97.1%**	6.39M
SketchAdapt [2]	95.0%	95.8%	~7M
SAPS [1]	93.2%	92.0%	5.73M

Trained and evaluated on the cleaned dataset
Using a beam size of 10
Evaluated on I/O pairs

[1] Bunel, Hausknecht, Devlin, Singh, Kohli 2018
[2] Nye, Hewitt, Tenenbaum, Solar-Lezama 2019

Performance

Model	Dev Accuracy	Test Accuracy	Parameters
Model_81 (No Token Pairing)	97.7%	97.1%	6.39M
Model_200	**99.0%**	**98.9%**	6.47M
SketchAdapt [2]	95.0%	95.8%	~7M
SAPS [1]	93.2%	92.0%	5.73M

Trained and evaluated on the cleaned dataset
Using a beam size of 10
Evaluated on I/O pairs

[1] Bunel, Hausknecht, Devlin, Singh, Kohli 2018
[2] Nye, Hewitt, Tenenbaum, Solar-Lezama 2019

Performance

Model	Dev Accuracy	Test Accuracy	Parameters
Model_81 (No Token Pairing)	97.7%	97.1%	6.39M
Model_200	**99.0%**	**98.9%**	6.47M
Model_300	99.0%	98.7%	6.52M
SketchAdapt [2]	95.0%	95.8%	~7M
SAPS [1]	93.2%	92.0%	5.73M

Trained and evaluated on the cleaned dataset
Using a beam size of 10
Evaluated on I/O pairs

[1] Bunel, Hausknecht, Devlin, Singh, Kohli 2018
[2] Nye, Hewitt, Tenenbaum, Solar-Lezama 2019

I/O Pair Evaluation vs Golden Program

Model	Dev Accuracy	Test Accuracy
Model_200 - I/O Pair Evaluation	**99.0%**	**98.9%**
Model_200 - Golden Program Evaluation	98.9%	98.5%

Trained and evaluated on the cleaned dataset
Using a beam size of 10

State of NPS

- ML Models can pretty reliably produce working code in a DSL
  - Can be procedurally converted to another language or machine code

- Code works but it is often slow
  - Many DSLs don’t take advantage of the RAM model of computing
  - ML models have yet to demonstrate algorithmic thinking

Conclusion

- We addressed issues with previous NPS models
  - Attention – difficulty of single sweep encoding
  - Syntax Layer – difficulty of learning language syntax
  - Token Pairing – difficulty of linear structure (novel)
- Our model significantly outperforms previous work
  - 98.9% vs 95.8%
- Writing fast code is still hard

Areas for Future Investigation

- Modifying the architecture to implicitly perform token pairing
  - Currently experimenting with using the Euler tour of the program tree
- Transfer learning – using pretrained encodings/weights from NLP models
- Algorithmic Thinking

Acknowledgements

- My Mentor, William Moses
- My Parents, Jun Wang and Ping Yan
- Prof. Slava Gerovitch
- Prof. Srini Devadas

Supplemental Slides

Bonus Experiment With Algorithmic Thinking

- Can a model predict the algorithms that a certain programming task may require?
- Scraped codeforces.com for programming problems tagged with the algorithms they involve
- After cleaning the dataset, we have about 5000 problems on which to train a classification model

Bonus Experiment With Algorithmic Thinking

Model	Test Accuracy (Macro F1 Score)
Bag of Words Feed-Forward Model	28.7%
Transformer Model [1]	24.0%

[1] Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin 2017