Lecture 32
Quantitative Reasoning and a Bayesian View of Synthesis

Sankha Narayan Guria

with slides from

Armando Solar-Lezama

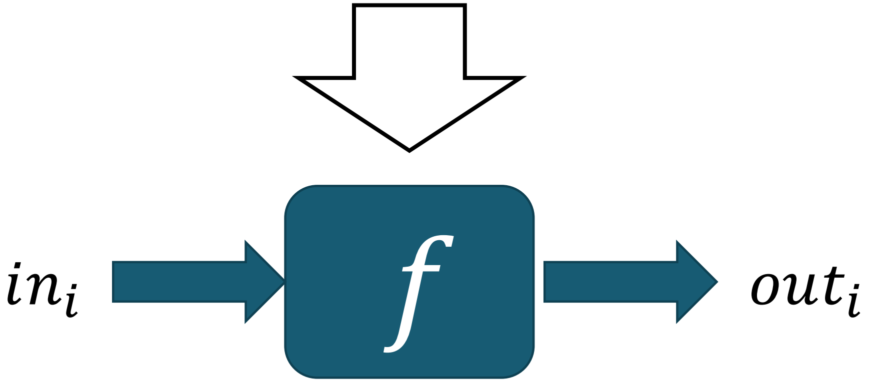
Bayes Theorem

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

$$P(A \cap B) = P(B \mid A)P(A) = P(A \mid B)P(B)$$

Programming by Example

$$[(in_0, out_0), (in_1, out_1), \ldots (in_k, out_k)]$$



Problem is hopelessly underspecified

- Many semantically distinct programs can satisfy the examples

Bayesian View of PBE

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

$$P(f \mid evidence) = \frac{P(evidence \mid f)P(f)}{P(evidence)}$$

I/O Examples

Bayesian View of PBE

Find the best f given the evidence

$$P(f \mid evidence) = \frac{P(evidence \mid f)P(f)}{P(evidence)}$$

Find the best f given the evidence

$$\operatorname*{argmax}_f P(f \mid evidence) = \operatorname*{argmax}_f \frac{P(evidence \mid f)P(f)}{P(evidence)}$$
$$= \operatorname*{argmax}_f P(evidence \mid f)P(f)$$

WARNING: $P(evidence)$ better not be zero!

$P(evidence \mid f)$

z is a normalization constant, crucial for making sure these are probabilities, but unimportant from the point of view of optimiziation.

Perfectly captured I/O examples

- $P(evidence \mid f\ ) = P([(in_i, out_i)]_i \mid f) = \begin{cases} 1/z & \forall_i\ f(in_i) = out_i \\ 0 & otherwise \end{cases}$
- With a uniform $P(f)$ this reduces to finding any function that works

$P(f)$

So far we have been using a uniform P

- $P(f) = \begin{cases} 1/Z & \text{if } f \text{ belongs to the space of programs} \\ 0 & \text{otherwise} \end{cases}$

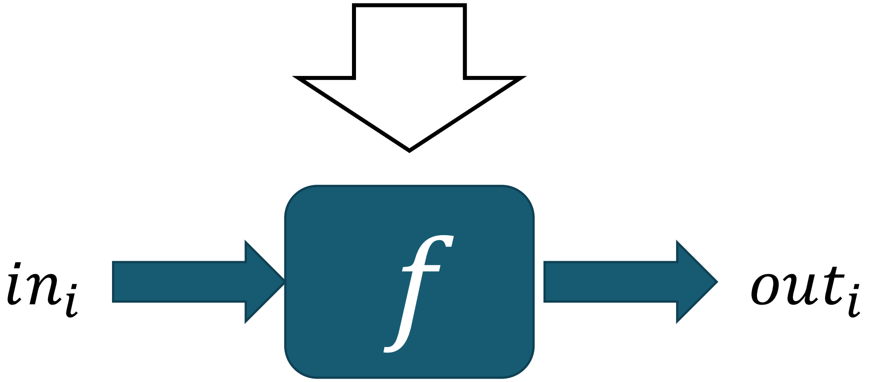
Shortest programs are better than longer programs

- $P(f) = \begin{cases} \frac{1}{Z} * e^{-len(f)} & \text{if } f \text{ belongs to the space of programs} \\ 0 & \text{otherwise} \end{cases}$

Could we learn $P(f)$?

Programming by Example

$$[(in_0, out_0), (in_1, out_1), \dots (in_k, out_k)]$$



Problem is hopelessly underspecified

- Many semantically distinct programs can satisfy the examples

$$P(f \mid [(in_i, out_i)]_i) \approx P_f(f) * P_{io}([(in_i, out_i)]_i \mid f)$$

$P(evidence \mid f)$

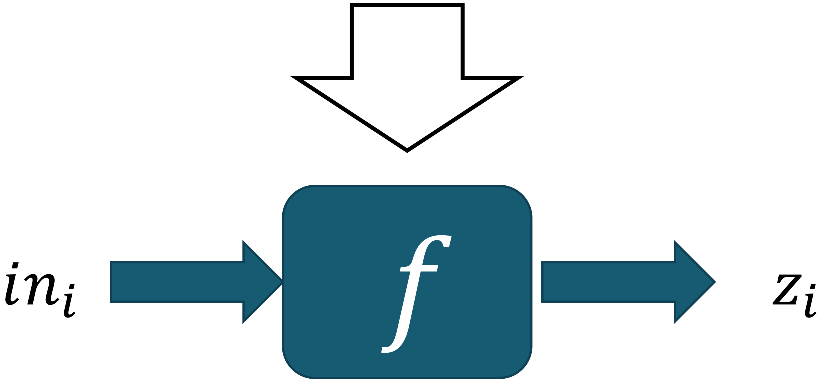
$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Imperfectly captured independent I/O examples

- $P(evidence \mid f\ ) = P([(in_i, out_i)]_i \mid f) = \prod_i P_{o|z}(out_i \mid f, in_i)P(in_i)$
- For the purposes of maximizing $P(f)$, $P(in_i)$ can be ignored if all inputs are equally likely

Learning from noisy data

$$[(in_0, out_0), (in_1, out_1), \ldots (in_k, out_k)]$$



Need to trade off quality of f against faithfulness to data

- This requires an error model

$$P(f \,|[(in_i, out_i)]_i) \approx P_f(f) * \prod_i P_{o|z}(out_i \mid f, in_i)$$

Off-by-one Errors

Suppose we know off-by-one errors are possible in the data

- $P_{o|z}(out_i \mid f, in_i) = \begin{cases} 0.5 & f(in_i) = out_i \\ 0.25 & f(in_i) = out_i \pm 1 \\ 0 & else \end{cases}$

If $p(f)$ is uniform, this reduces to

- "Discard programs that are more than one-off on any input"
- "From the remaining programs, select the one that matches the most examples"

Off-by-one Errors

Suppose we know off-by-one errors are possible in the data but others are possible as well.

- $P_{o|z}(out_i \mid f, in_i) = \begin{cases} \frac{1}{Z}0.5 & f(in_i) = out_i \\ \frac{1}{z}0.25 & f(in_i) = out_i \pm 1 \\ \epsilon & else \end{cases}$

Non-uniform $P(f)$

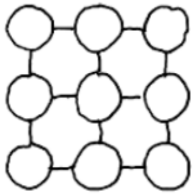
Trade off $P_{o|z}(out_i \mid f, in_i)$ against $P(f)$

- A solution that misses more outputs may still be preferable if it has much higher probability

Learning to Infer Graphics Programs from Hand-Drawn Images

with Kevin Ellis, Daniel Ritchie, Josh Tenenbaum

From images to programs



Hand Drawn Figure

NN + Search

```
Circle(5,8)
Circle(2,8)
Circle(8,11)
Line(2,9, 2,10)
Circle(8,8)
Line(3,8, 4,8)
Line(3,11, 4,11)
Line(8,9, 8,10)
Circle(5,14)
```

... etc. ...; 21 lines

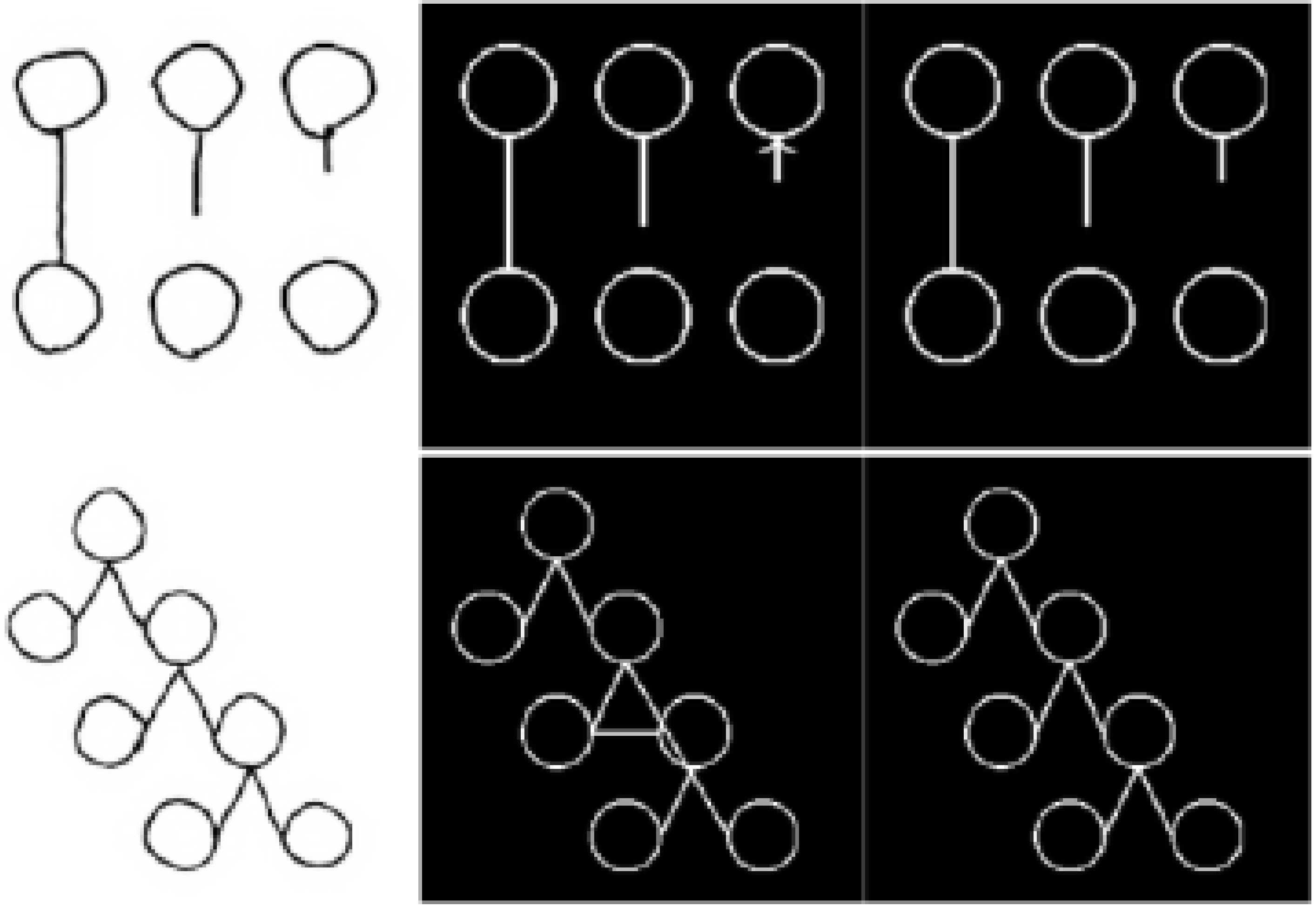
Description of
elements in the drawing

Synthesis

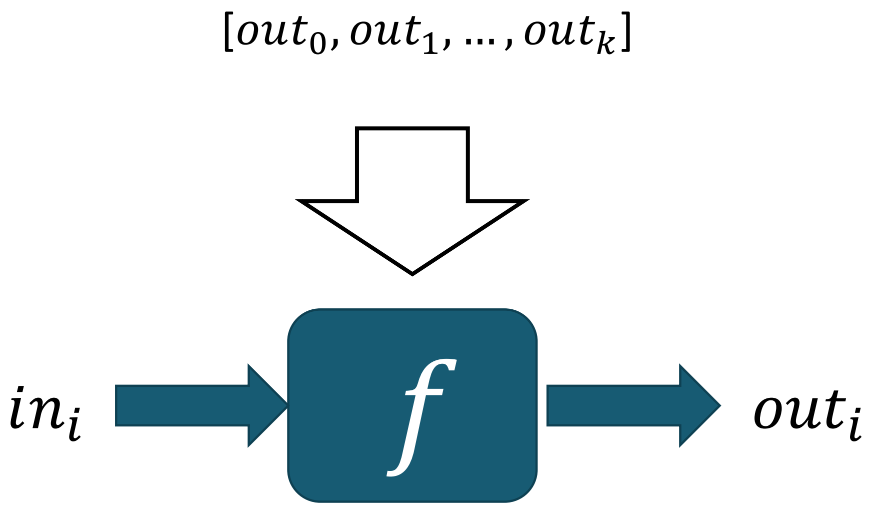
```
for(i<3)
 for(j<3)
  if(j>0)
   line(-3*j+8,-3*i+7,
        -3*j+9,-3*i+7)
  line(-3*i+7,-3*j+8,
       -3*i+7,-3*j+9)
  circle(-3*j+7,-3*i+7)
```

Program representation
of drawing

Why? Correcting errors in perception



Unsupervised learning



This is hopelessly underspecified

- Can we identify the process that generated the sequence?

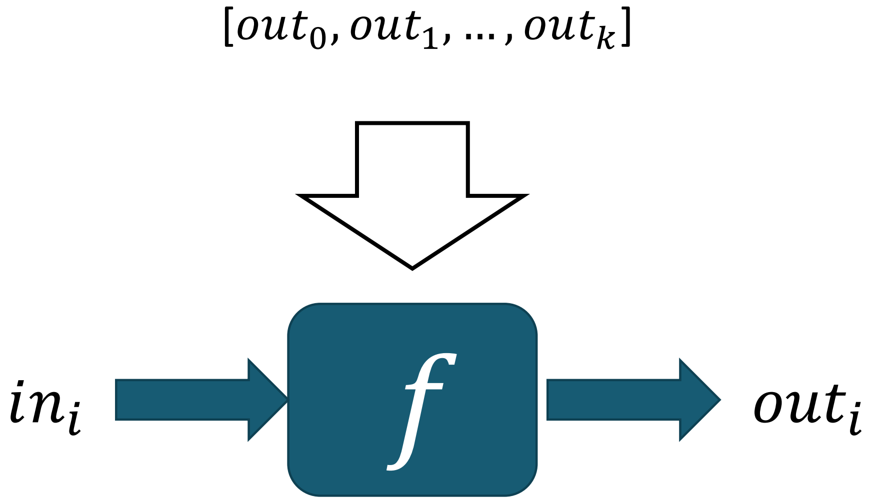
Unsupervized learning

$$P(f, [in_i] \mid [out_i]) = \frac{P([out_i] \mid f, [in_i])P(f, [in_i])}{P([out_i])}$$

Assuming independence: $P([out_i] \mid f, [in_i]) = \Pi_i \, P(out_i \mid f, in_i)$

$$P(f, [in_i]) = P(f) * \Pi_i \, P(in_i)$$

Unsupervised learning



This is hopelessly underspecified

- Can we identify the process that generated the sequence?

$$P(f, [in_i]_i | [out_i]_i) \approx P_f(f) * \prod_i P_{o|z}(out_i \mid f, in_i) * P_{in}(in_i)$$

To marginalize or not to Marginalize

$$P(f, [in_i]_i | [out_i]_i)$$

Probability that a given function and inputs were the cause for an observed series of outputs

$$\sum_{[in_i]_i} P(f, [in_i]_i | [out_i]_i) P([in_i]_i)$$

Probability that a given function is consistent with the observed outputs

Which of the two functions above should you be optimizing?

- Formulation on the left is easier to solve for
  - especially with symbolic methods

Maximum Likelihood vs Sampling

Often your goal is to find the most likely f

- $\max_f P_f(f \mid \ldots)$

For some situations, sampling from $P_f(f \mid \ldots)$ is better

- The most likely is not necessarily the one you want
- E.g. in PBE the function the user has in mind may not be the "best"

Isn't there a whole field looking into this?

Machine learning has been studying these problems for a while

What we bring to the table:

- Flexible spaces of functions
- Complex distributions
- Powerful symbolic search techniques

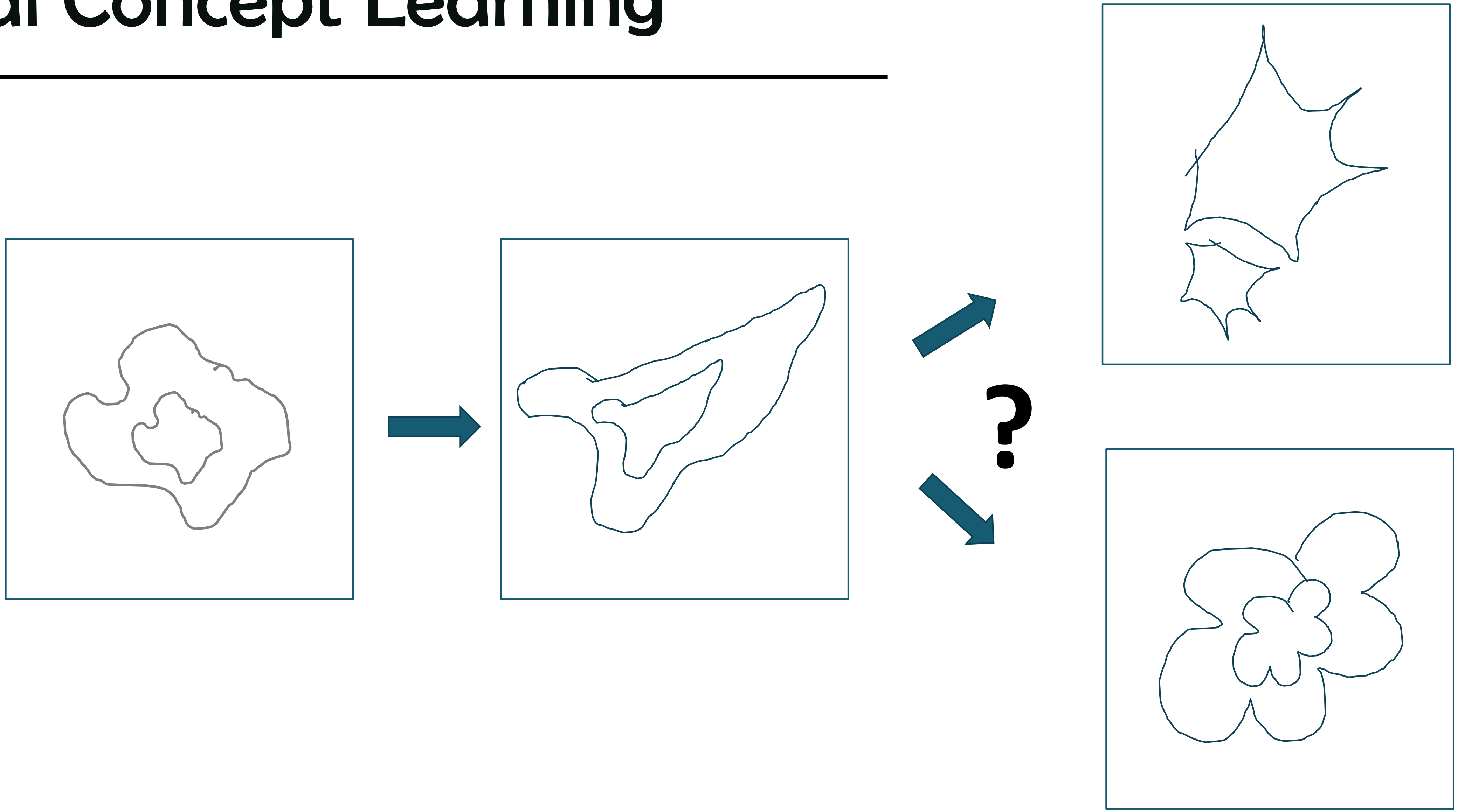
Maximum Likelihood

Optimization problem

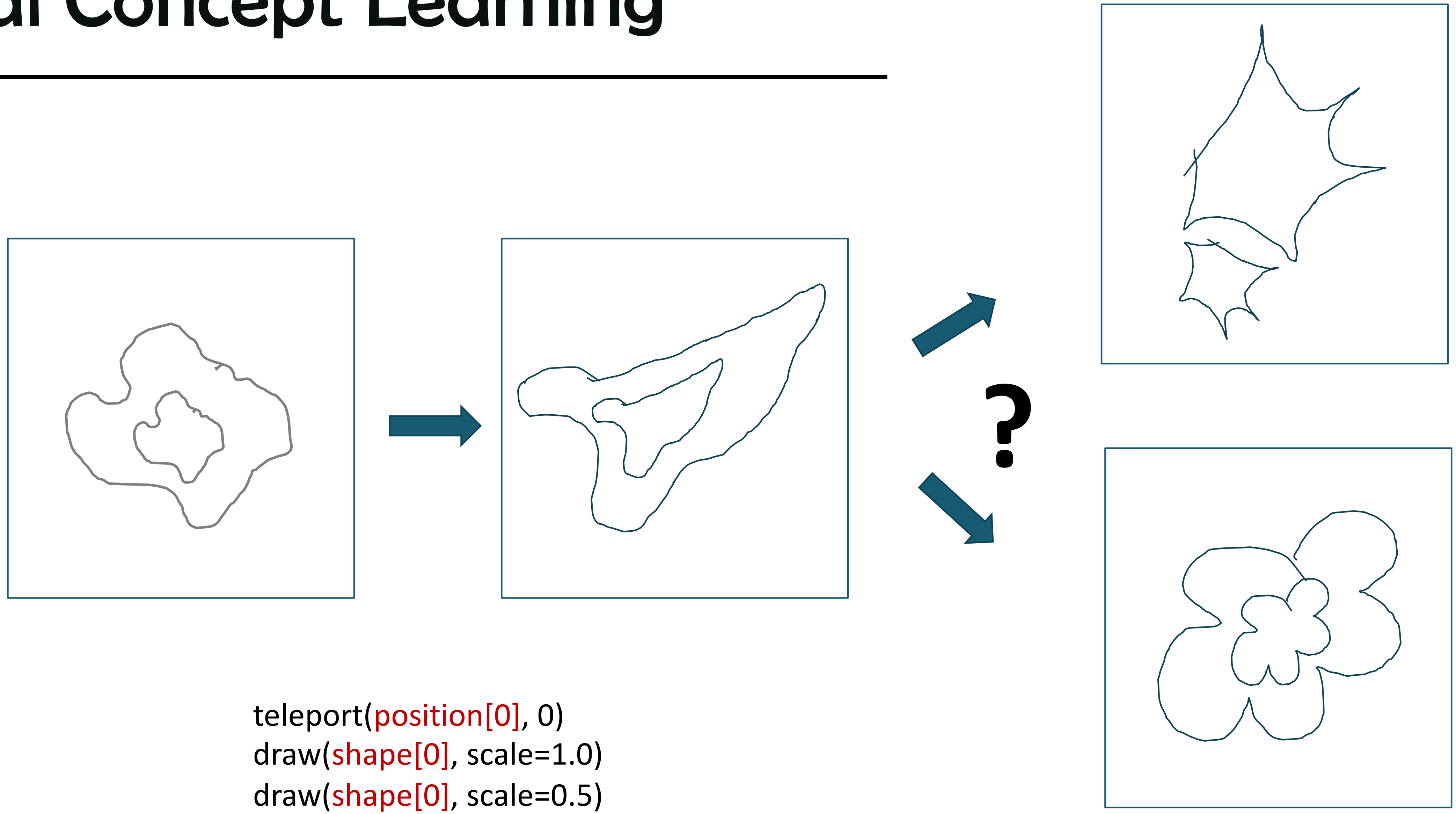
- $P_f(f) * \prod_i P_{o|z}(out_i \mid f(in_i)\ ) * P_{in}(in_i)$
- $\log\left(P_f(f)\right) + \sum_i \log(P_{o|z}(out_i \mid f(in_i)\ )) + \log(P_{in}(in_i))$

Easy to encode into SMT

Visual Concept Learning



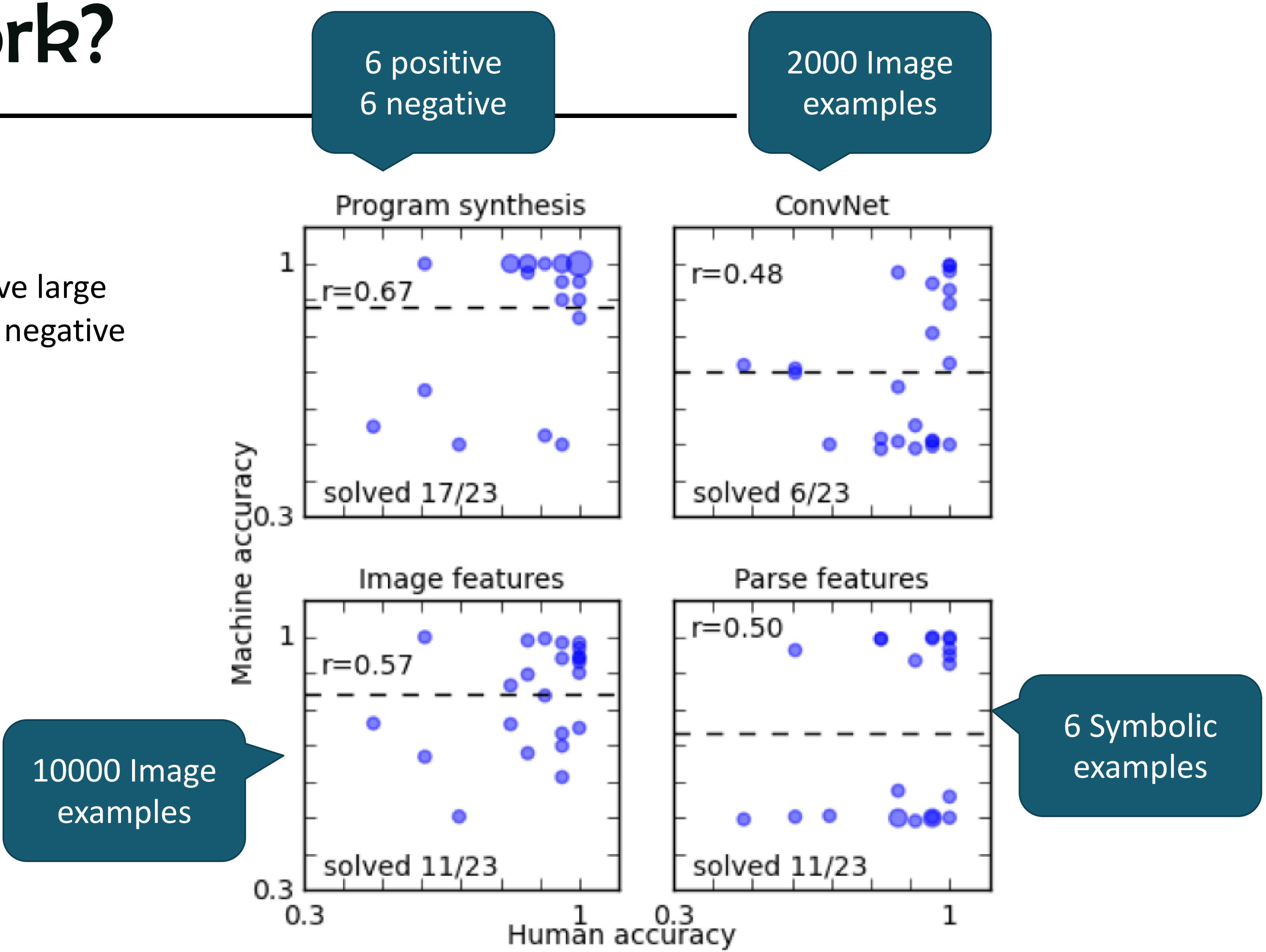
Visual Concept Learning




```
teleport(position[0], 0)
draw(shape[0], scale=1.0)
draw(shape[0], scale=0.5)
```

Does it work?

23 Challenge problems
For each problem we have large numbers of positive and negative examples.



Morphological Rule Learning

Learn rules for constructing different tenses of a word

Lexeme	Present	Past	3rd Sing. Pres.	Past Part.	Prog.
style	staɪl	staɪld	staɪlz	staɪld	staɪlɪŋ
run	rʌn	ræn	rʌnz	rʌn	rʌnɪŋ
subscribe	səbskraɪb	səbskraɪbd	səbskraɪbz	səbskraɪbd	səbskraɪbɪŋ
rack	ræk	rækt	ræks	rækt	rækɪŋ

$$f : \langle stem\,, tense \rangle \rightarrow word$$

Unsupervised learning

$$f : \langle stem\ , tense \rangle \rightarrow word$$

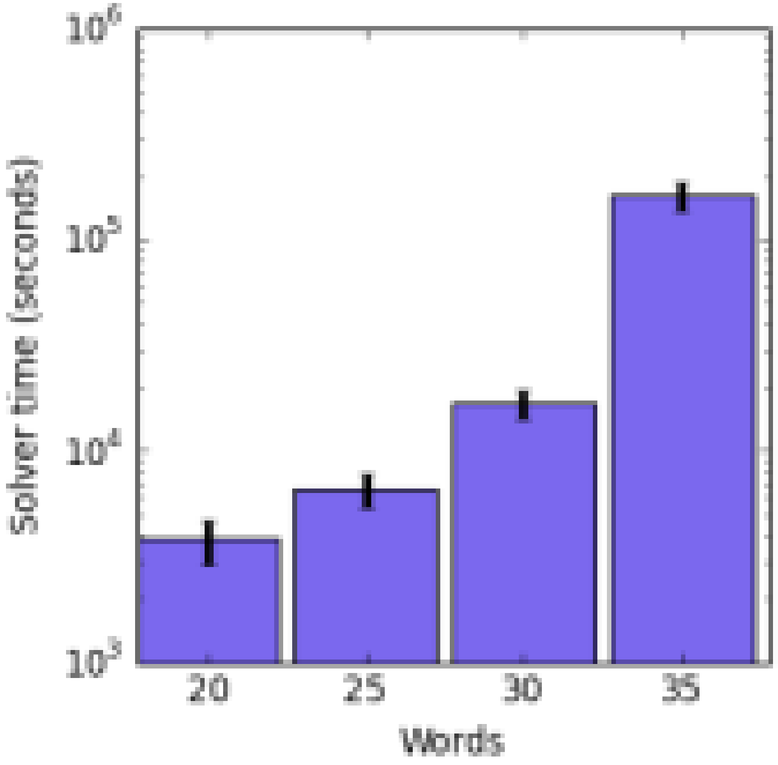
The stem is unknown

- We can define $P(stem \mid lexeme)$

Challenge:

- Need to learn from large amounts of noisy data
- Synthesis time grows quickly with number of words

Synthesis time



RANSAC

Random Sample Consensus

Developed by Martin Fischler and Robert Bolles in 1980

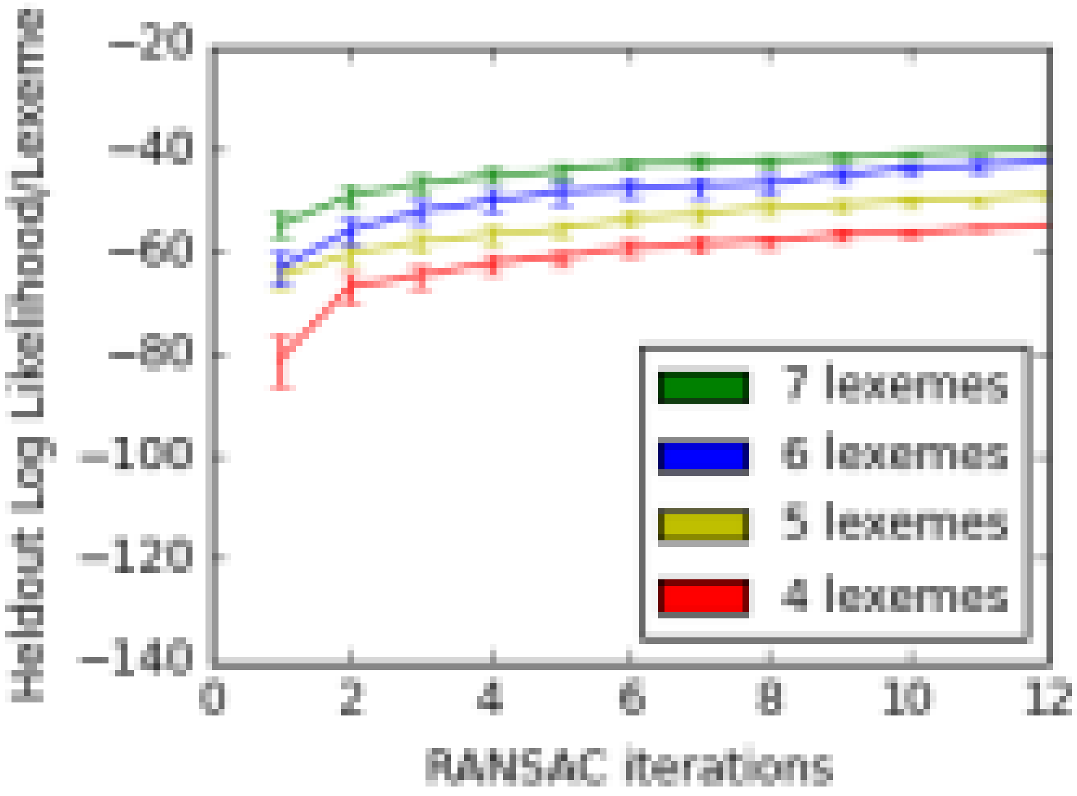
Designed to cope with gross errors in data

RANSAC

Key ideas:

- Pick small random samples from the noisy data
- Learn from the samples
- Incorporate other samples consistent with the learned model
- Repeat

Precision with # of iterations



State of the art

	Synthesis	Morfessor
Error Rate	3.16%	16.43%

Caveats: This is not a fair comparison

- Synthesis based solution has much more domain knowledge
- But, that’s part of the point

Learning program distributions

Program Correction

```
def evaluatePoly ( x0 , x1 ) :
  x2 = 0
  for x3 in range ( len ( x0 ) - 1) :
   x2 = x2 + x0 [ x3 ] * ( x1 ** x3 )
 return x2
```

→

```
def evaluatePoly ( x0 , x1 ) :
  x2 = 0
  for x3 in range ( len ( x0 ) ) :
   x2 = x2 + x0 [ x3 ] * ( x1 ** x3 )
 return x2
```

Skipgrams

A skipgram is a natural language structure consisting of a sequence of words with a gap in the middle

Example:

I am going $\square$ the store

Program statements as skipgrams

```
def evaluatePoly ( x0 , x1 ) :
  x2 = 0

    x2 = x2 + x0 [ x3 ] * ( x1 ** x3 )
  return x2
```

Correction as a regeneration proccess

```
def evaluatePoly ( x0 , x1 ) :
  x2 = 0
  for x3 in range ( len ( x0 ) - 1) :
   x2 = x2 + x0 [ x3 ] * ( x1 ** x3 )
 return x2
```

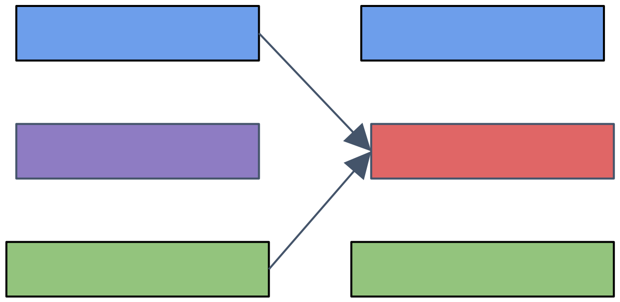
```
def evaluatePoly ( x0 , x1 ) :
  x2 = 0

   x2 = x2 + x0 [ x3 ] * ( x1 ** x3 )
 return x2
```

```
def evaluatePoly ( x0 , x1 ) :
  x2 = 0
  for x3 in range ( len ( x0 ) ) :
   x2 = x2 + x0 [ x3 ] * ( x1 ** x3 )
 return x2
```

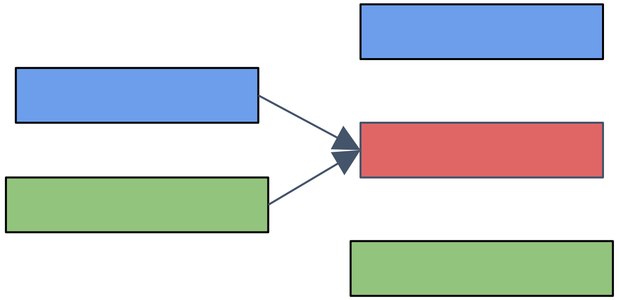
Replacement Insertion Deletion

Our simple correction scheme is flexible:

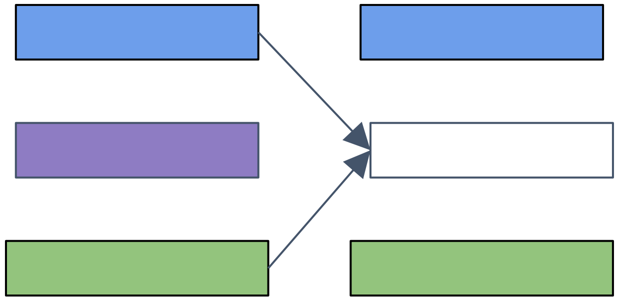
Replacement



Insertion

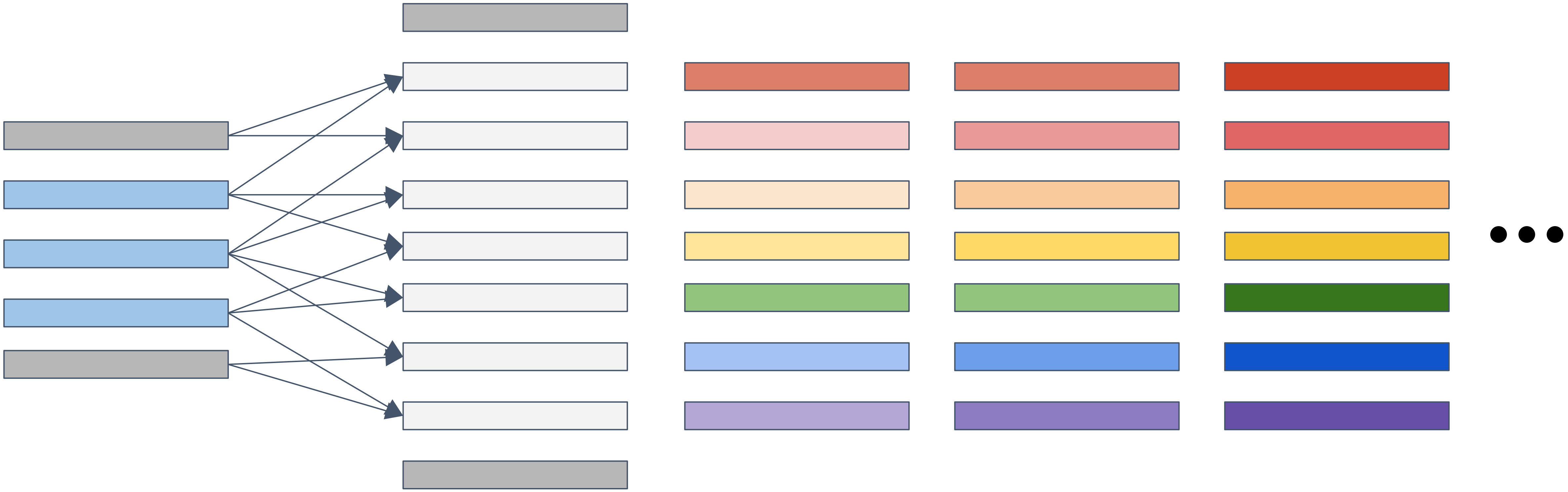


Deletion



Space of edits

We consider all possible replacement, insertions, and deletions given our skipgram context taken in 1 step simultaneously:

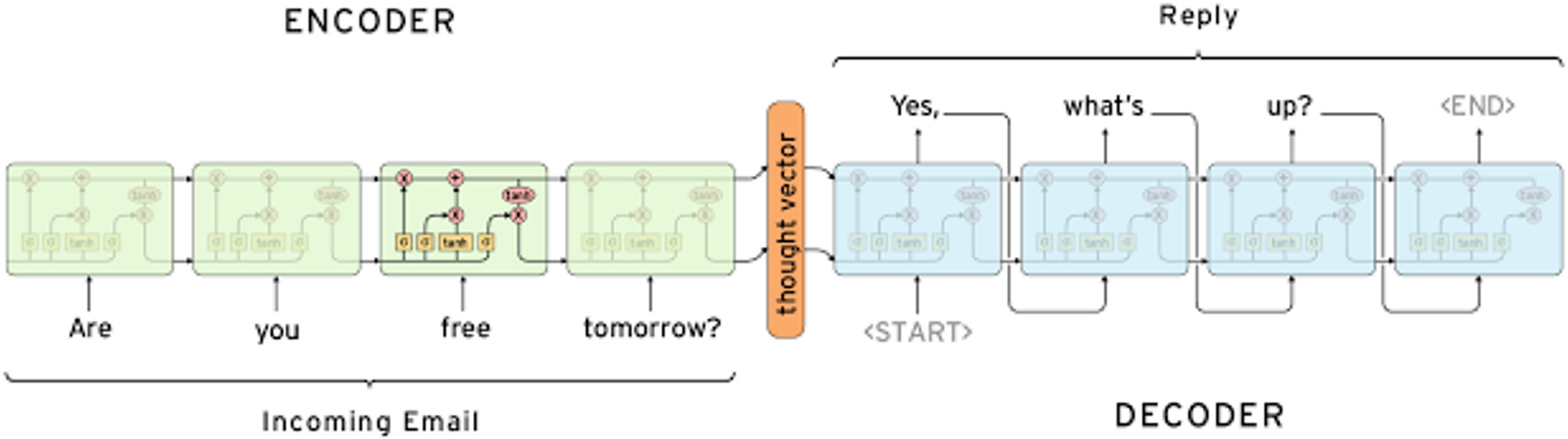


Original Program

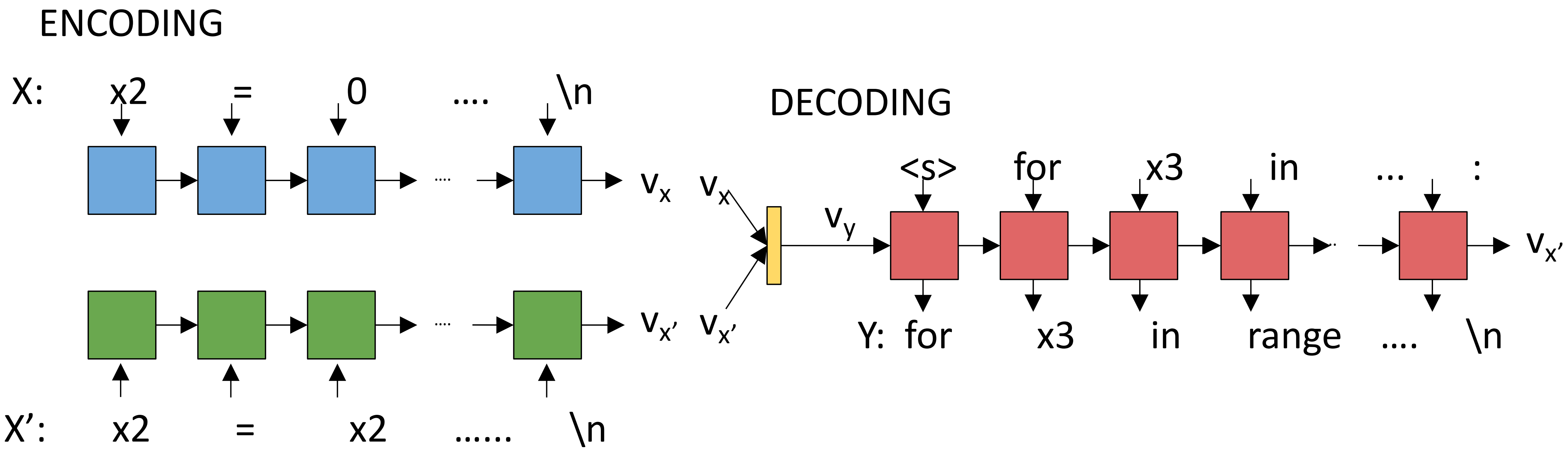
Possible Edits

Candidates for each Edit

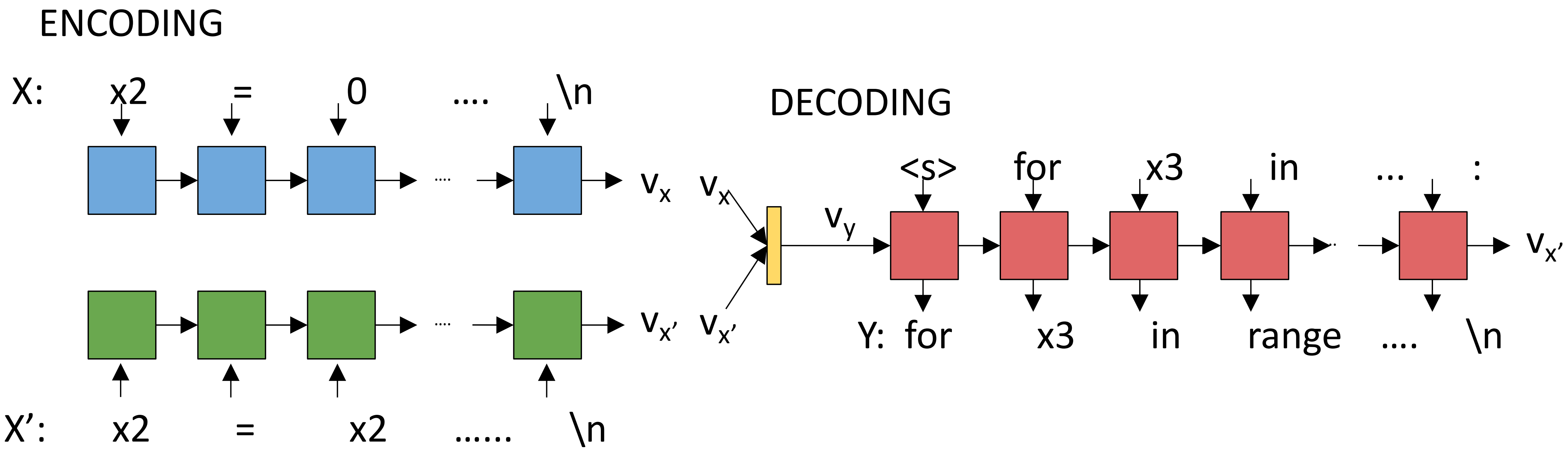
Seq2seq modeling



Modifying Seq2Seq for candidate generation



Probability of Candidate



Pr(for x3 in … \n | X, X') = Pr(for x3 in … \n | v_y) = Pr(for | v_y, <s>) * Pr(x3 | v_y, <s>, for) * …

Experiments

Benchmarks	training_raw_n	trianing_filtered_n	testing_n	filterd_score	raw_score
computeDeriv	1256	912 (0.726)	295	23/142 = 0.162	23/295 = 0.078
computeRoot	1611	1185 (0.735)	101	12/58 = 0.207	12/101 = 0.119
evaluatePoly	2304	1881 (0.816)	117	38/54 = 0.703	38/117 = 0.324
oddTuples	8720	6903 (0.791)	2009	491/957 = 0.513	491/2009 = 0.244