Lecture 34
Learning
Complex Distributions

Sankha Narayan Guria

With slides from

Armando Solar-Lezama

n-gram models

The big brown bear scares the children with its roar

$P(scares\ |bear, brown)$

Probability of a word depends on the previous n words

Represented with a table: $P(w_i\ |w_{i-1}, w_{i-2}, \dots, w_{i-n})$

Bigger n makes more accurate, but also more difficult to learn, requires much bigger table

Downsides

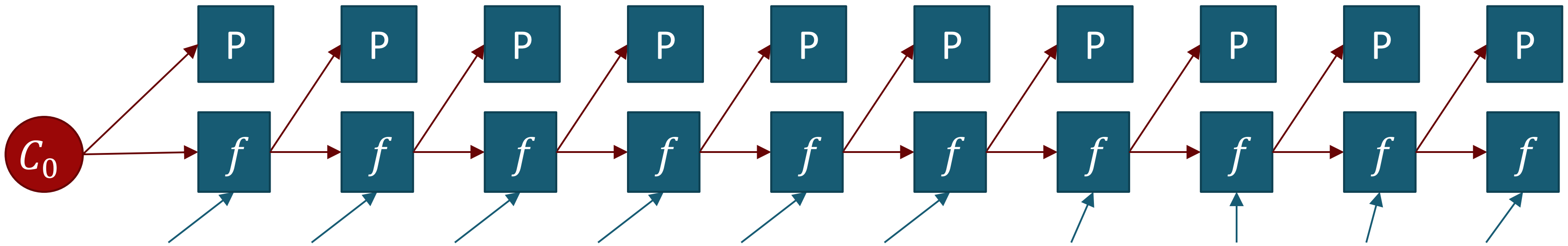
- some words require more context than others
- some words carry very little information

Recurrent models

Key idea: let the system figure out how to construct its own context

Now need to learn two interrelated functions

- $P(word_i \mid context_{i-1})$
- $context_i = f(word_i, context_{i-1})$

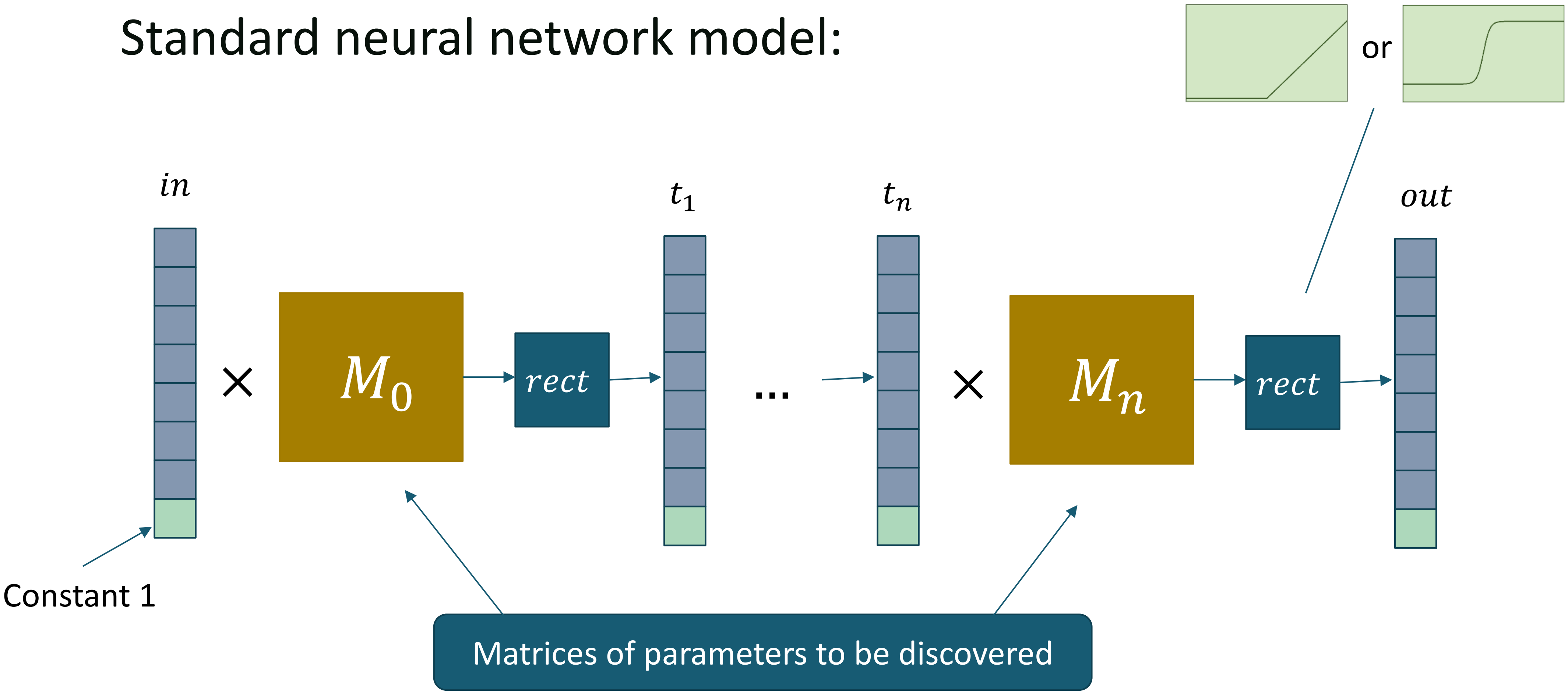


The big brown bear scares the children with its roar

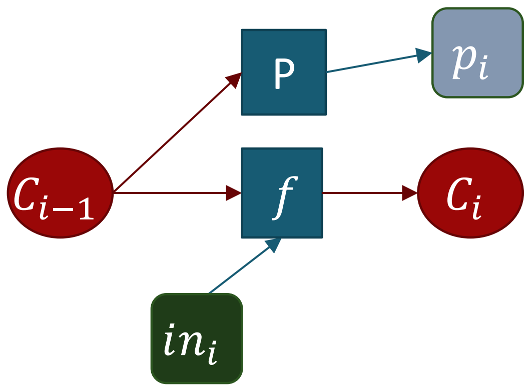
Neural Networks

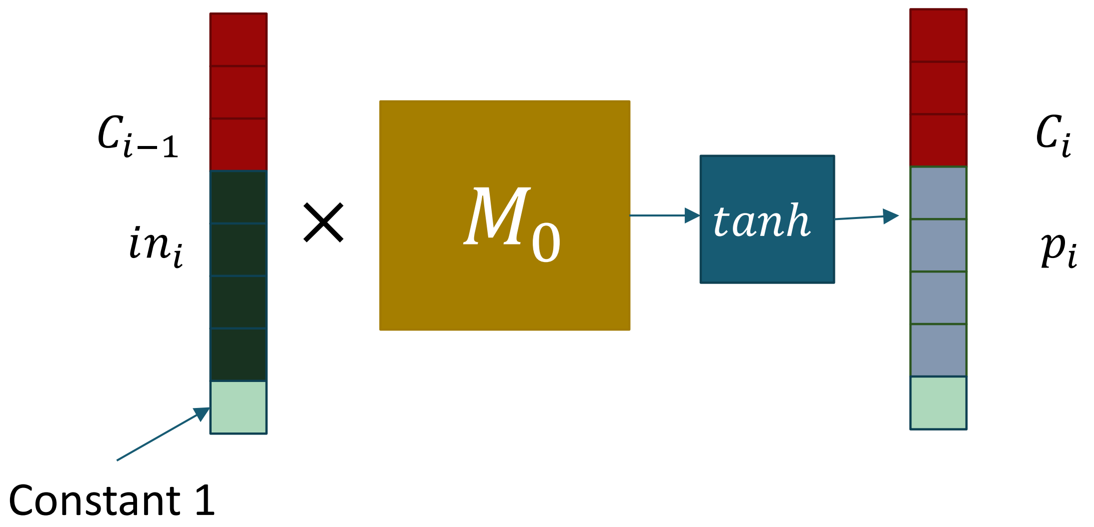
Most popular way of representing recurrent models

Standard neural network model:

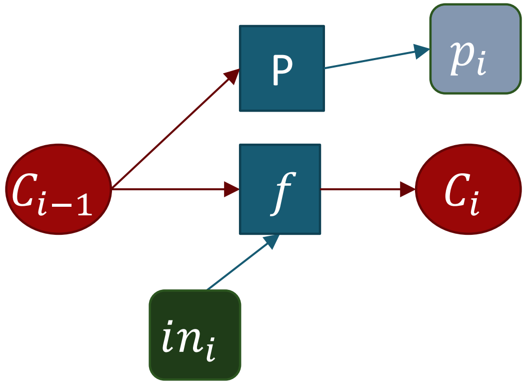


Simple recurrent neural network

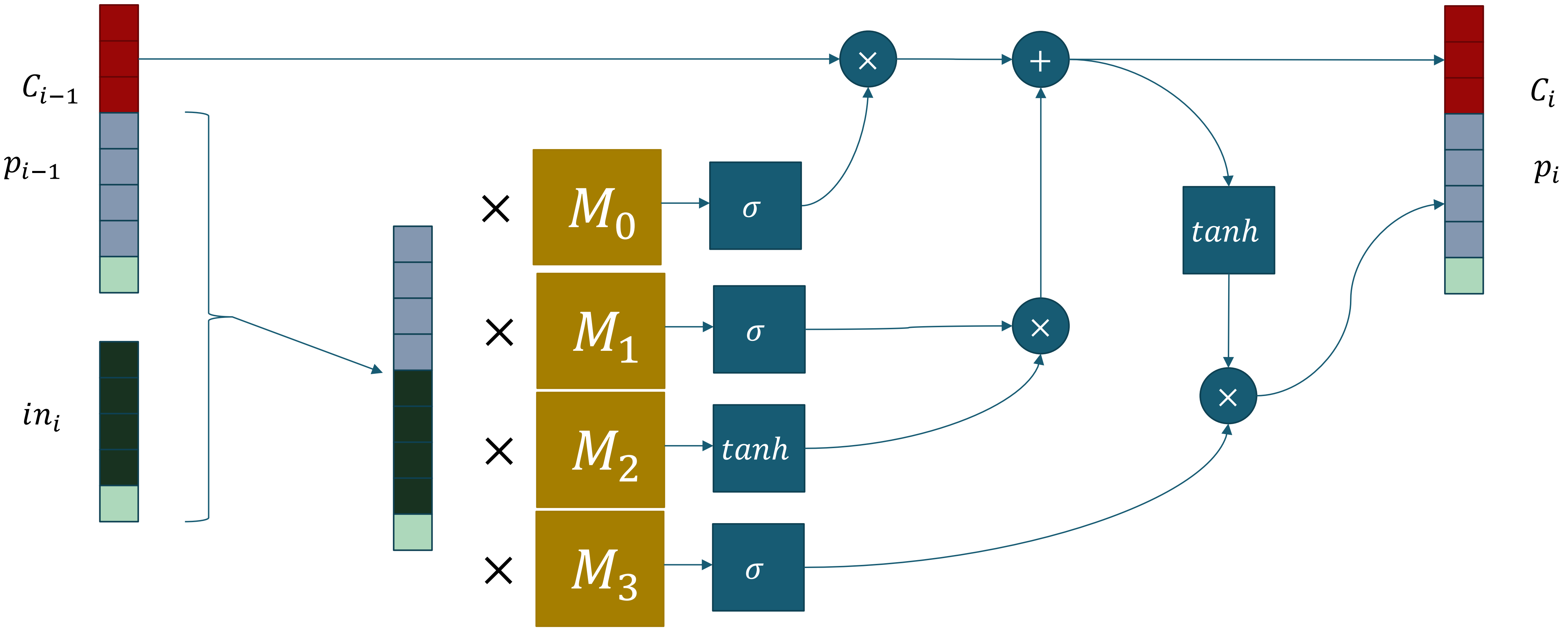




LSTM

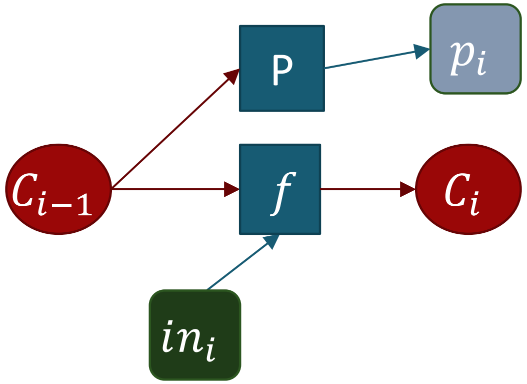


P
p_i
C_{i-1}
f
C_i
in_i

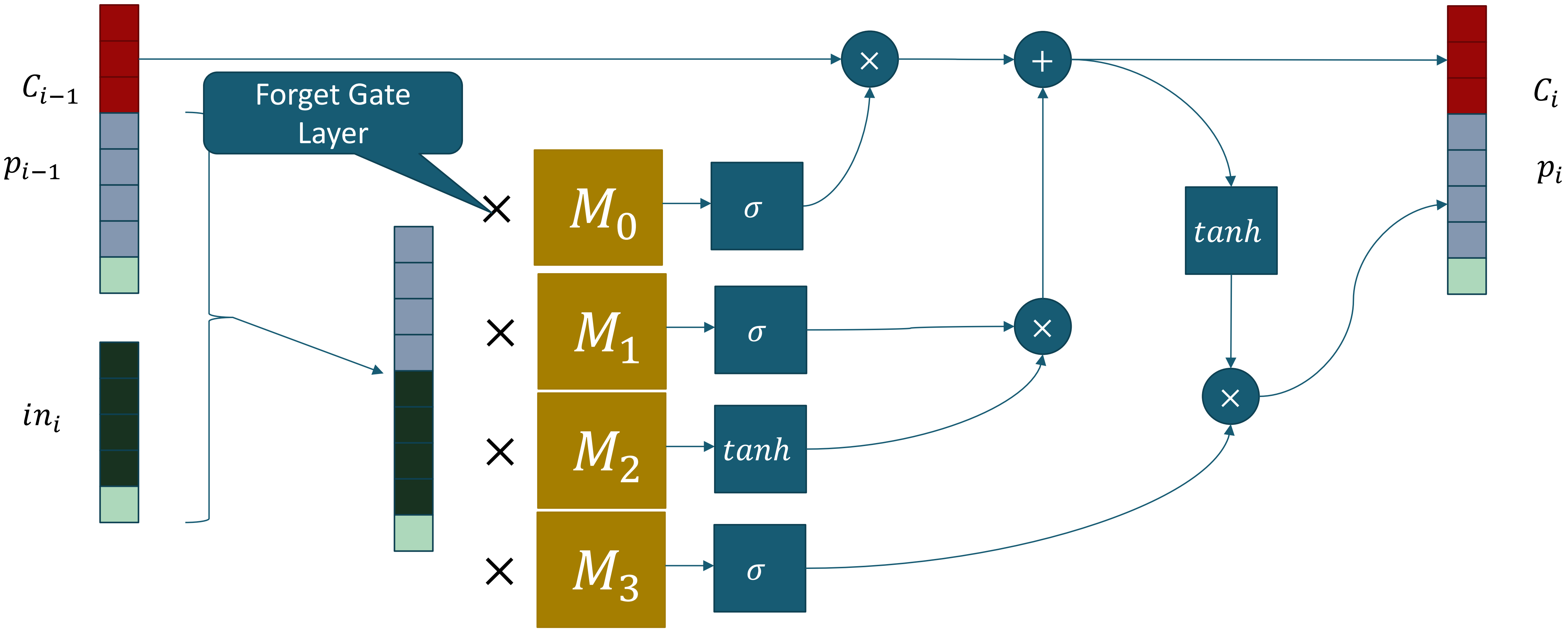


C_{i-1}
p_{i-1}
in_i
$\times$
M_0
M_1
M_2
M_3
σ
σ
tanh
σ
$\times$
$+$
$\times$
tanh
$\times$
C_i
p_i

LSTM

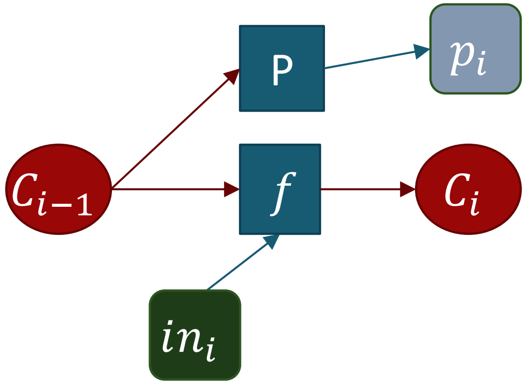


P
p_i
C_{i-1}
f
C_i
in_i

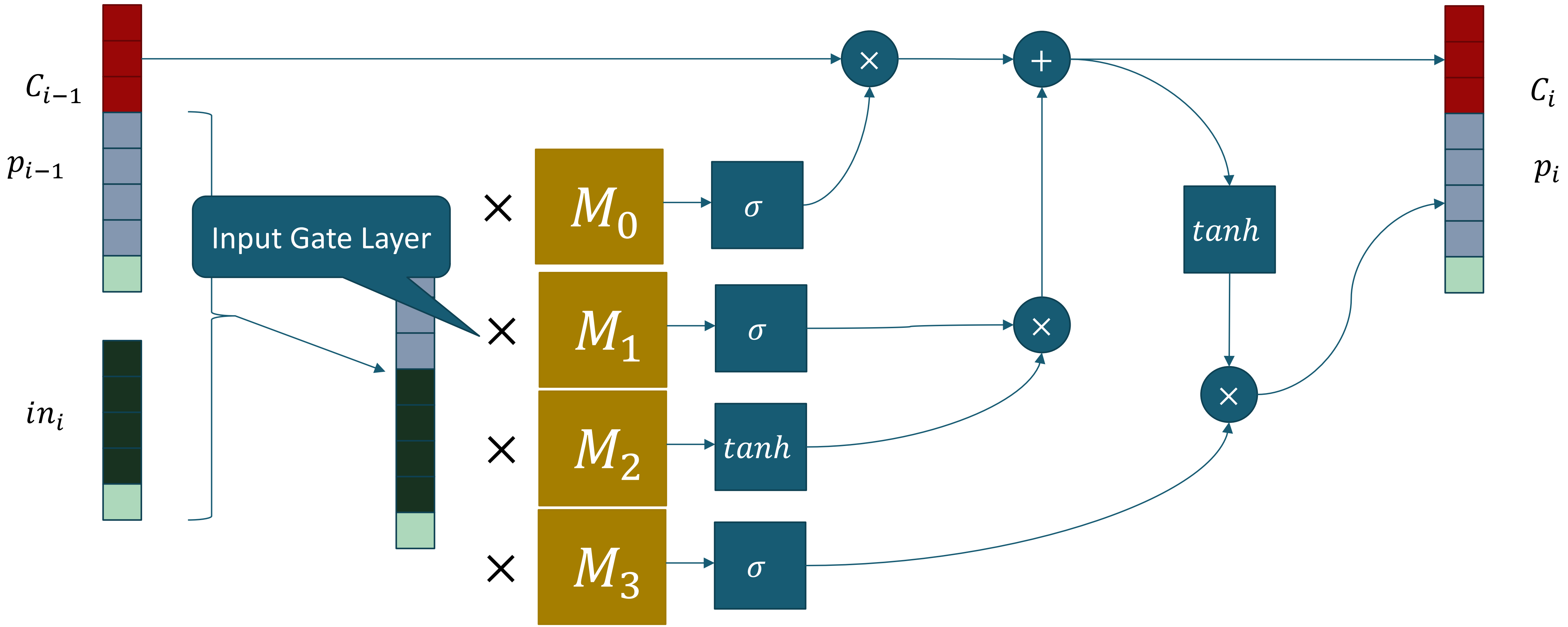


C_{i-1}
p_{i-1}
in_i
Forget Gate Layer
M_0
M_1
M_2
M_3
σ
σ
tanh
σ
tanh
C_i
p_i

LSTM

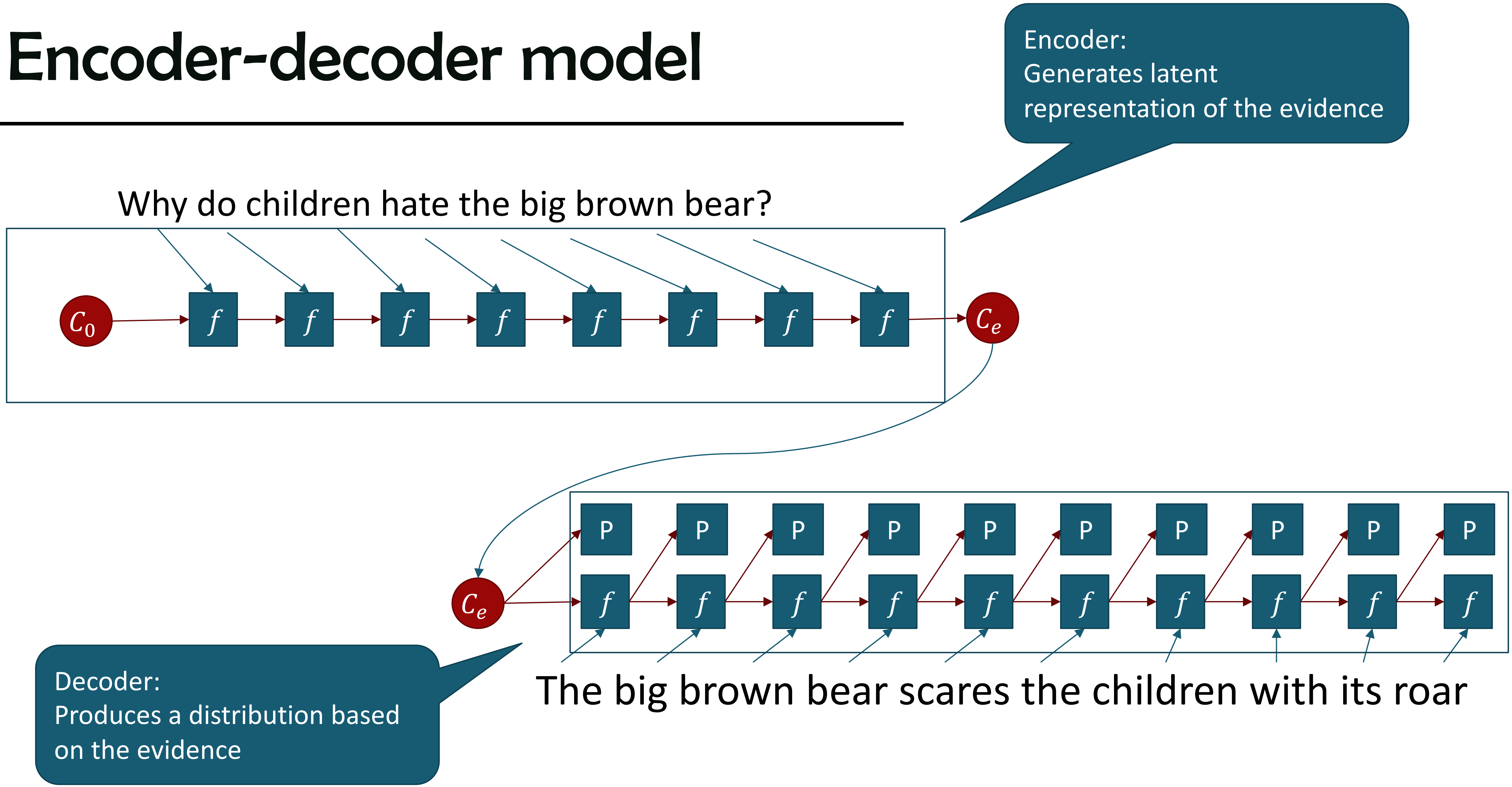


P
p_i
C_{i-1}
f
C_i
in_i



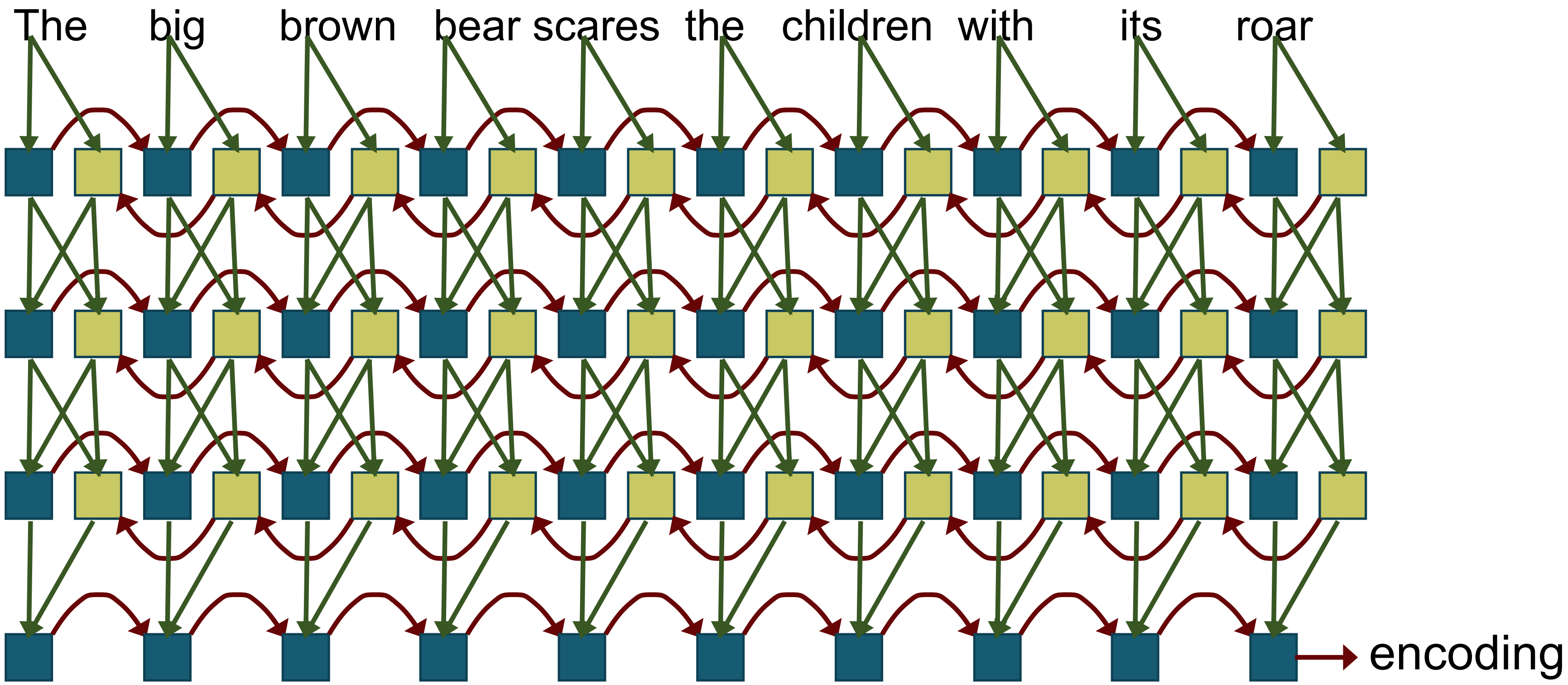
C_{i-1}
p_{i-1}
in_i
Input Gate Layer
$\times$ M_0
$\times$ M_1
$\times$ M_2
$\times$ M_3
σ
σ
$tanh$
σ
$\times$
$+$
$\times$
$tanh$
$\times$
C_i
p_i

Encoder-decoder model



Multilayer and bidirectional models

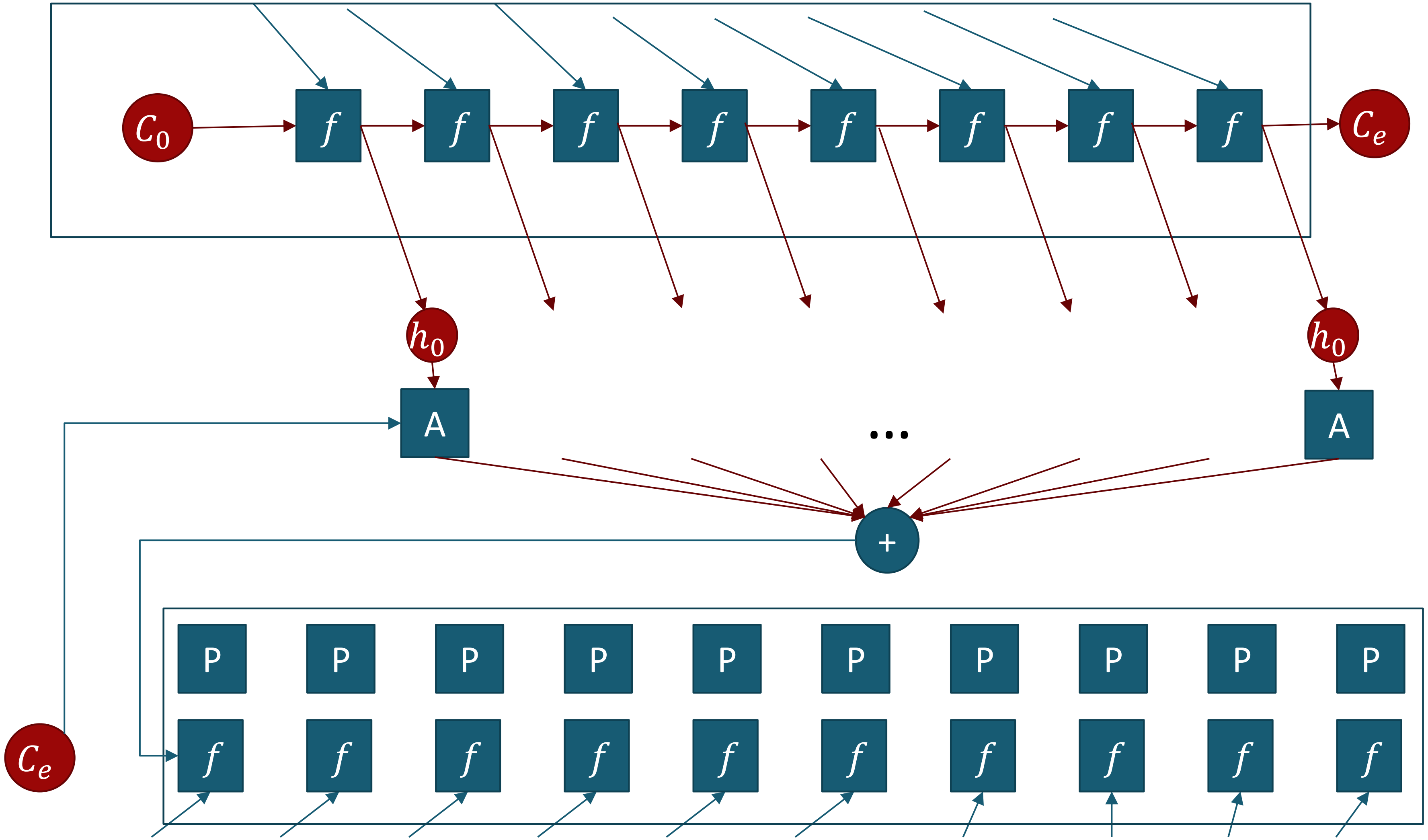
Bidirectional networks propagate information back-to-front as well as front-to-back. They can also have more than one layer.



Attention

Key idea: Summarizing into a single vector is a big bottleneck.
Every output should have direct access to the whole input

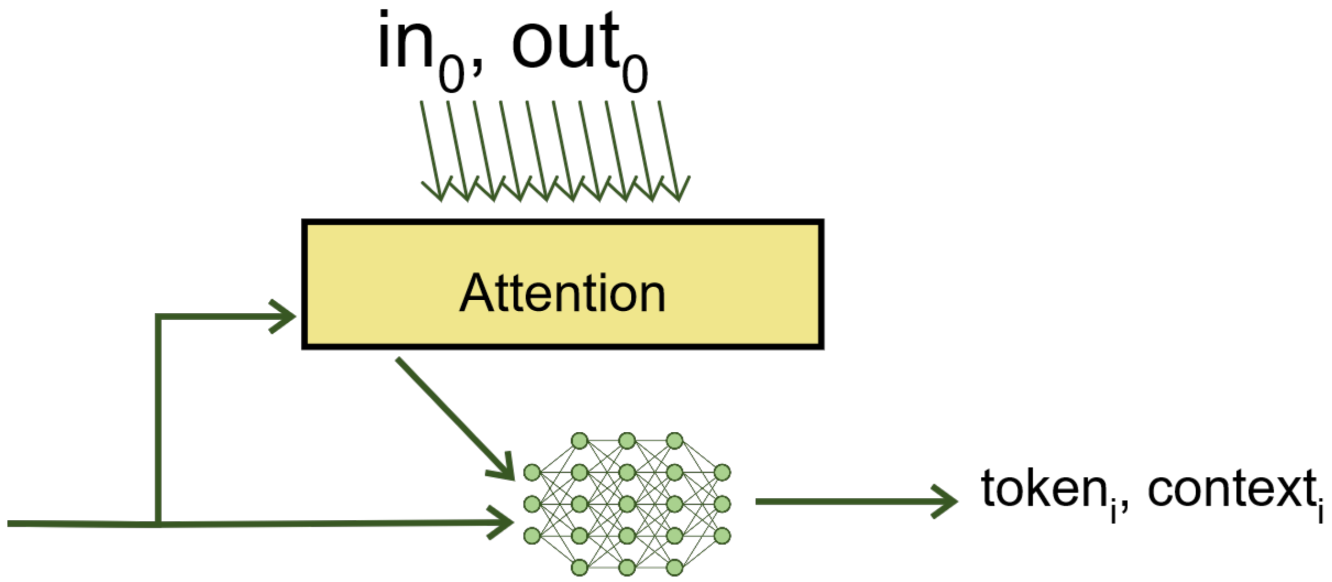
Attention



RobustFill

Single example case

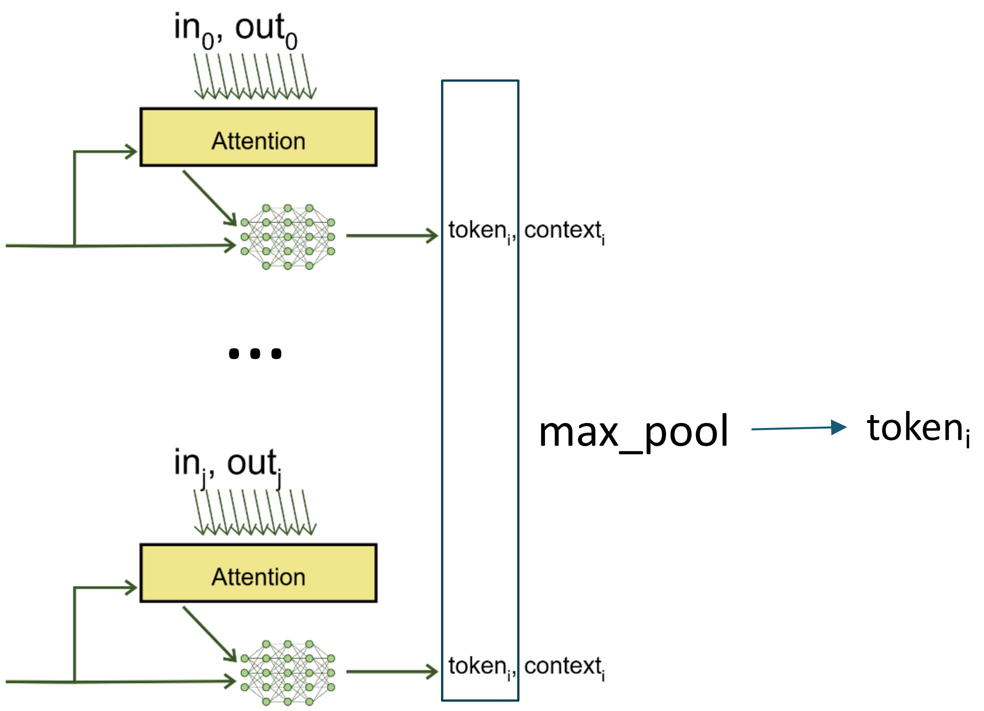
- Attend over the input/output to produce each token



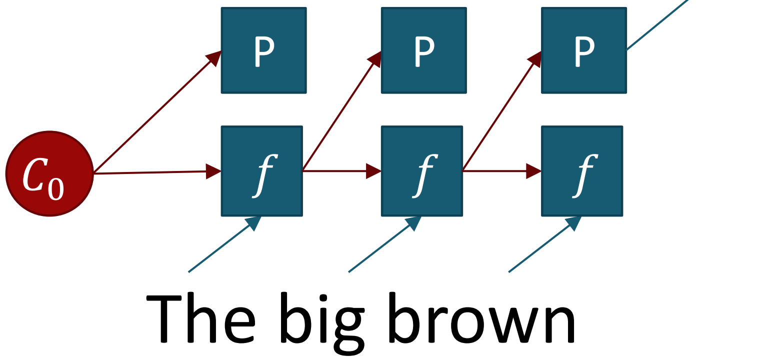
RobustFill

Multiple example case

- Compute each example independently
- pool over the result to decide which token to output



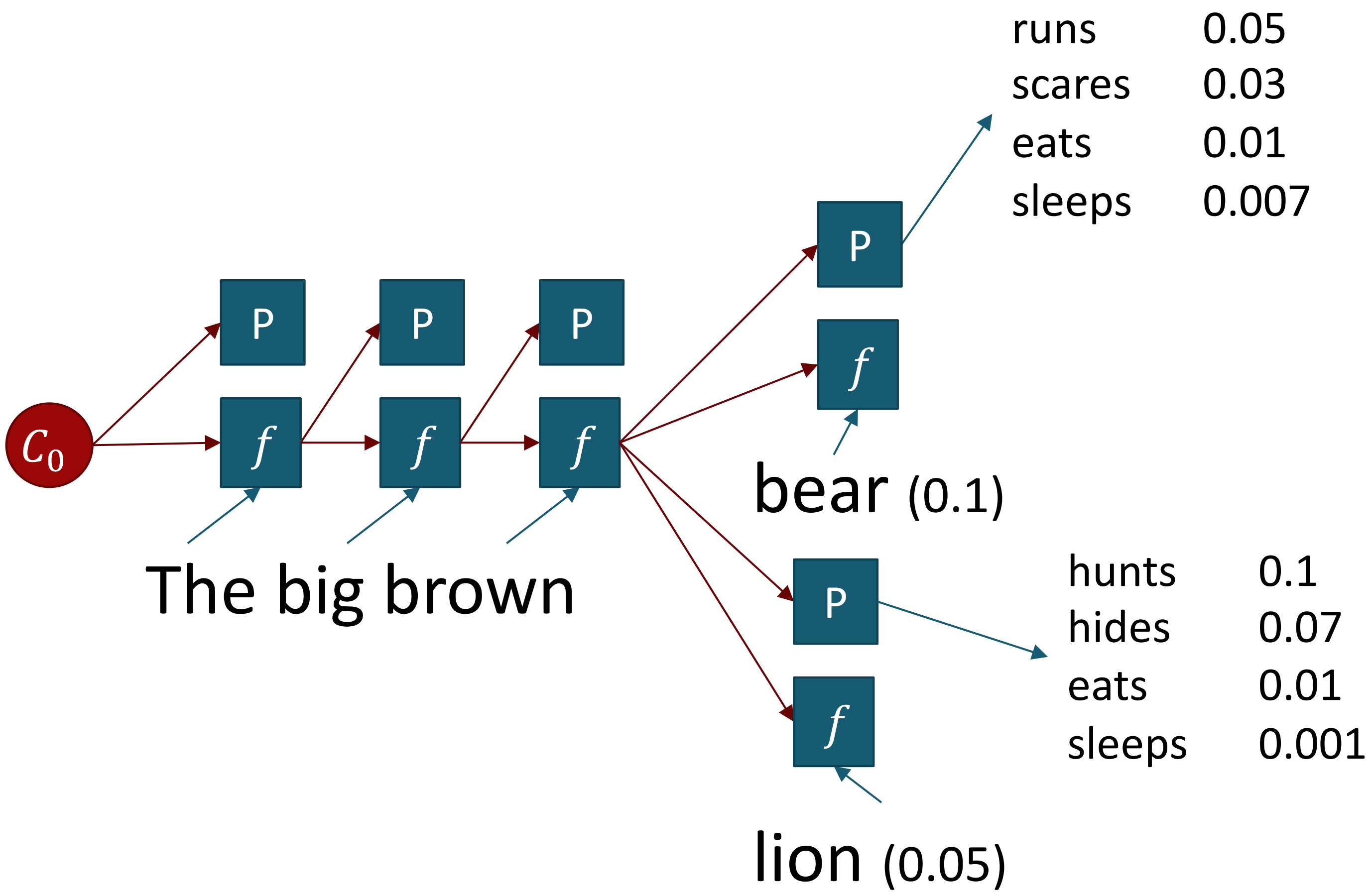
Beam Search



bear	0.1
lion	0.05
fox	0.01
cat	0.001

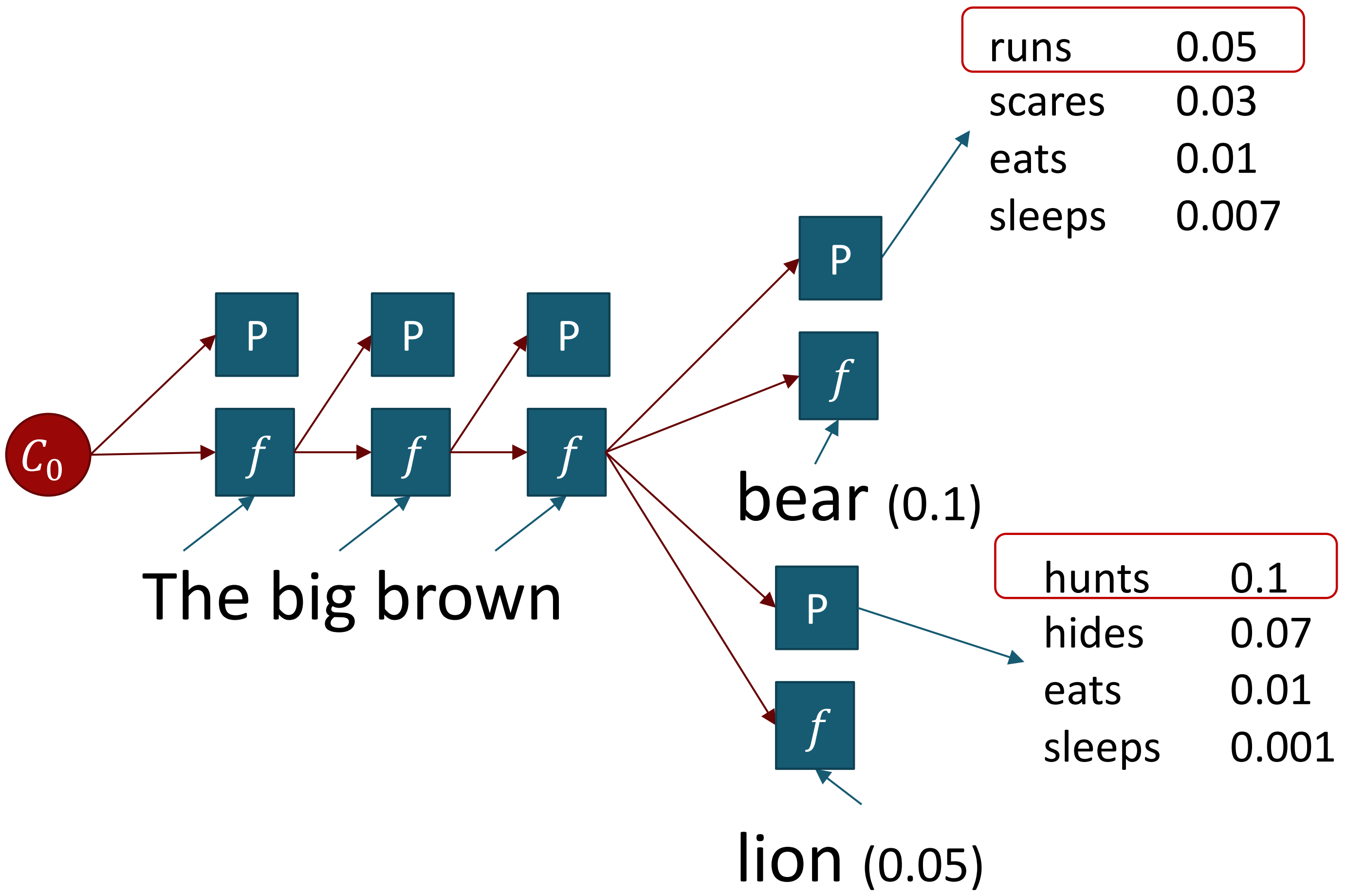
At each step pick the K most likely successors

Beam Search



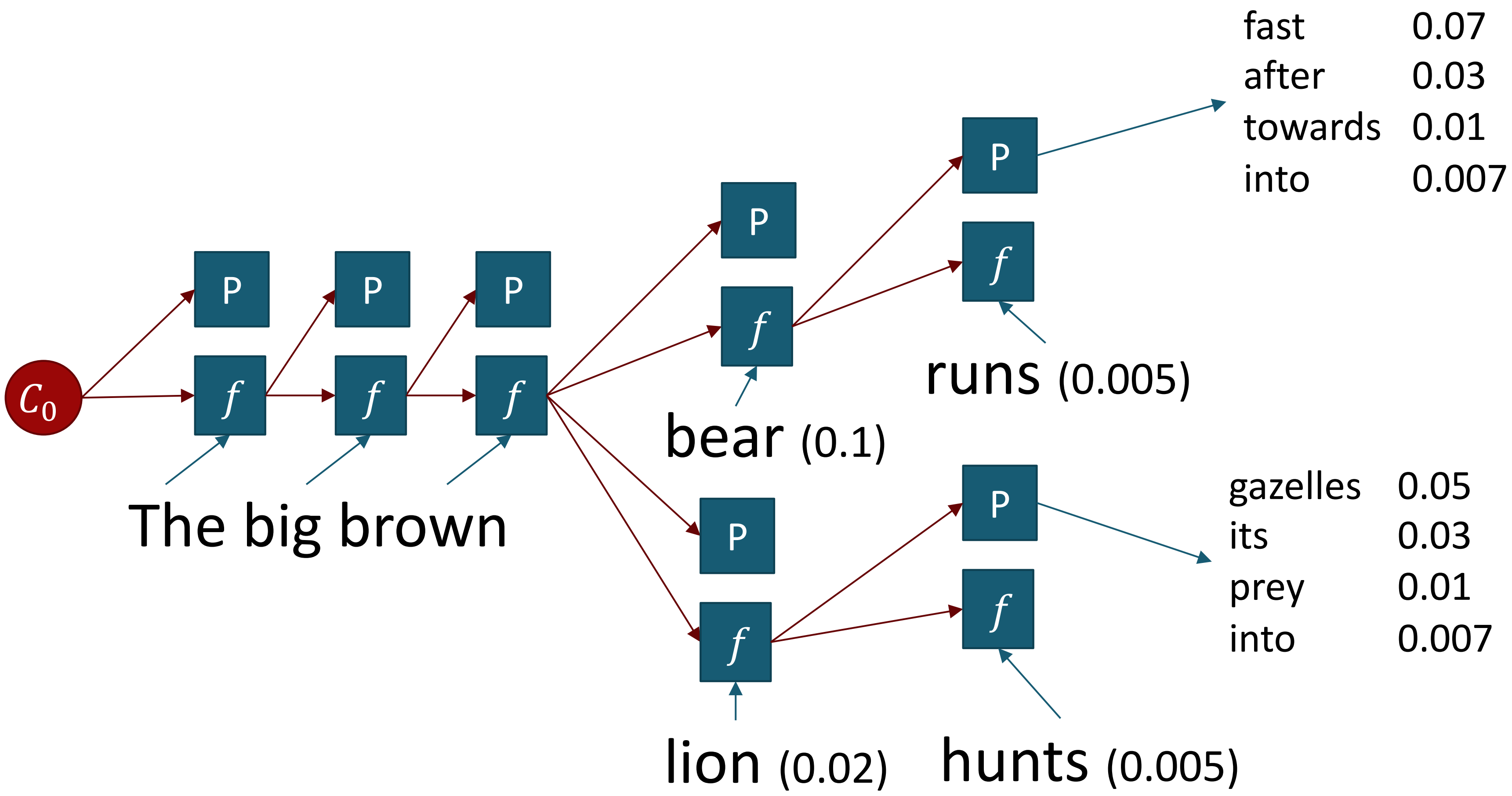
runs 0.05
scares 0.03
eats 0.01
sleeps 0.007
P
P
P
P
c_0
f
f
f
f
bear (0.1)
The big brown
P
hunts 0.1
hides 0.07
eats 0.01
sleeps 0.001
f
lion (0.05)

Beam Search



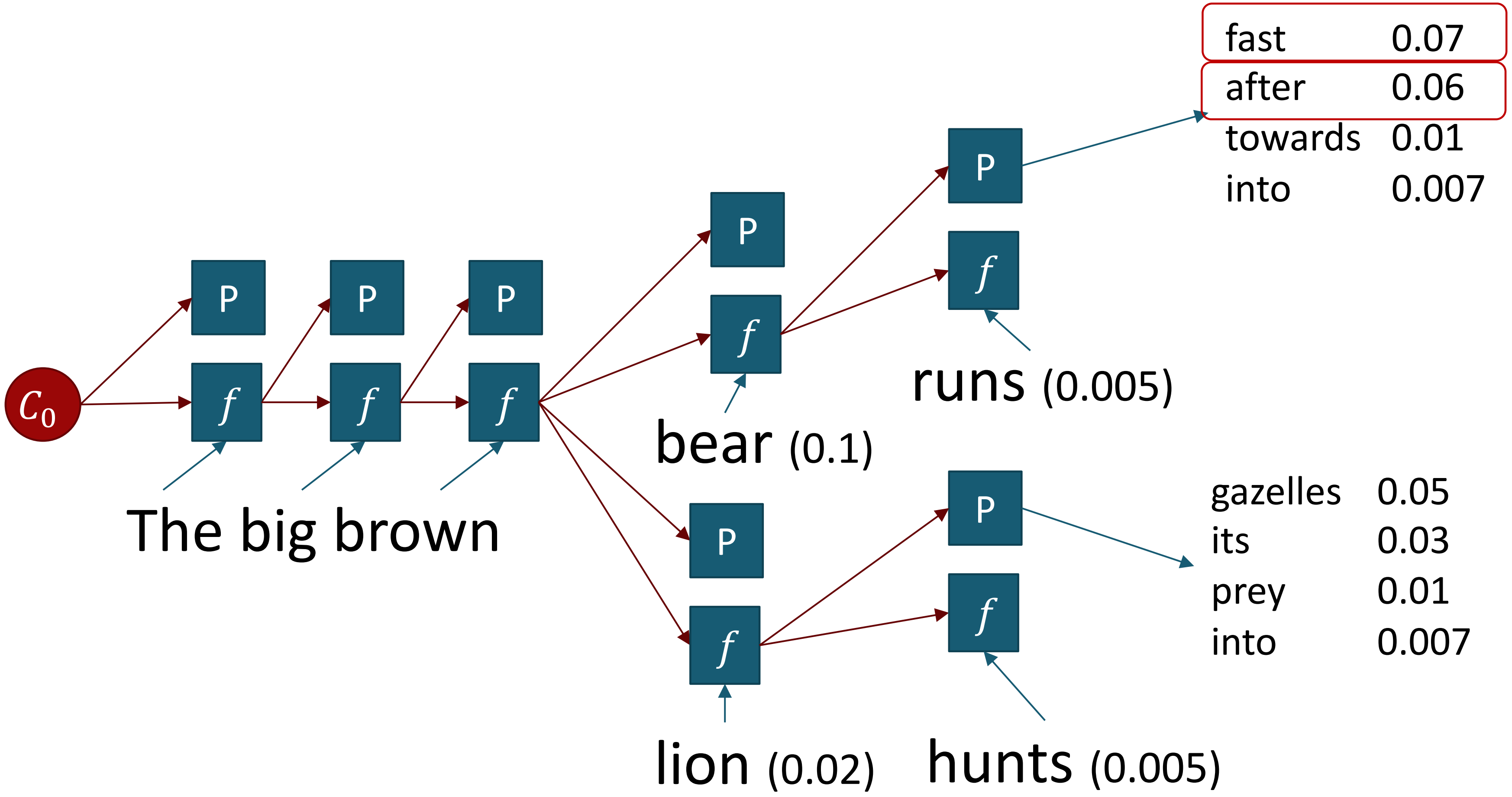
runs 0.05
scares 0.03
eats 0.01
sleeps 0.007
P
P P P
f
c_0
f f f
bear (0.1)
The big brown
hunts 0.1
P
hides 0.07
eats 0.01
f
sleeps 0.001
lion (0.05)

Beam Search



c_0
P
f
The big brown
bear (0.1)
lion (0.02)
runs (0.005)
hunts (0.005)
fast 0.07
after 0.03
towards 0.01
into 0.007
gazelles 0.05
its 0.03
prey 0.01
into 0.007

Beam Search



fast 0.07
after 0.06
towards 0.01
into 0.007
P
P
f
runs (0.005)
P
f
bear (0.1)
c_0
P
P
P
f
f
f
The big brown
P
f
lion (0.02)
P
f
hunts (0.005)
gazelles 0.05
its 0.03
prey 0.01
into 0.007

Beam search for programs

Similar to top-down search

- Same strategies for pruning the search space apply

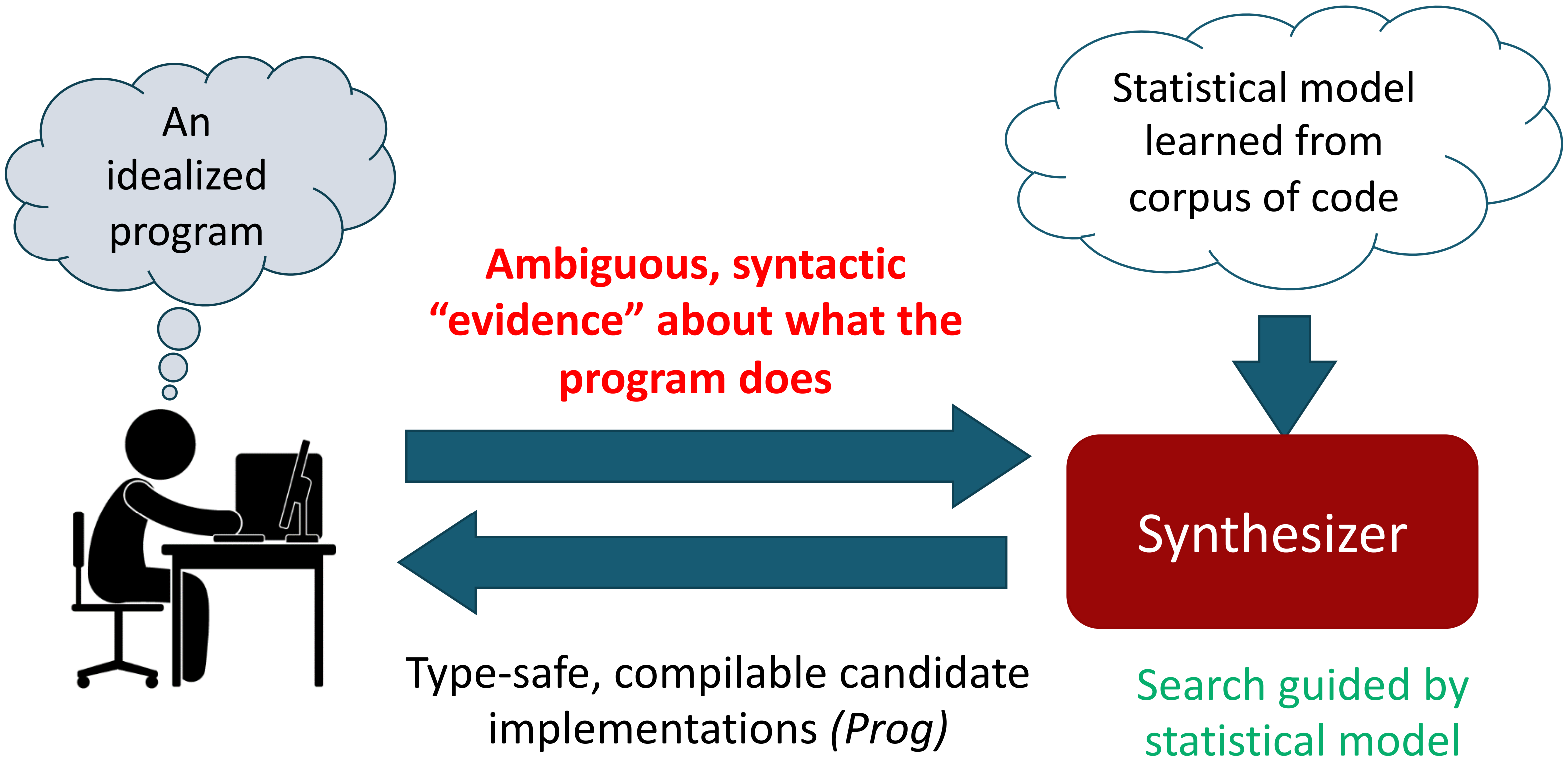
Bayesian View of PBE

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

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

Can we learn $P(f|evidence)$ directly?

Synthesis from ambiguous evidence



Neural Sketch Learning for Conditional Program Generation. Murali, Qi, Chaudhuri, and Jermaine. ICLR 2018 (oral presentation).

The Bayou system: Example

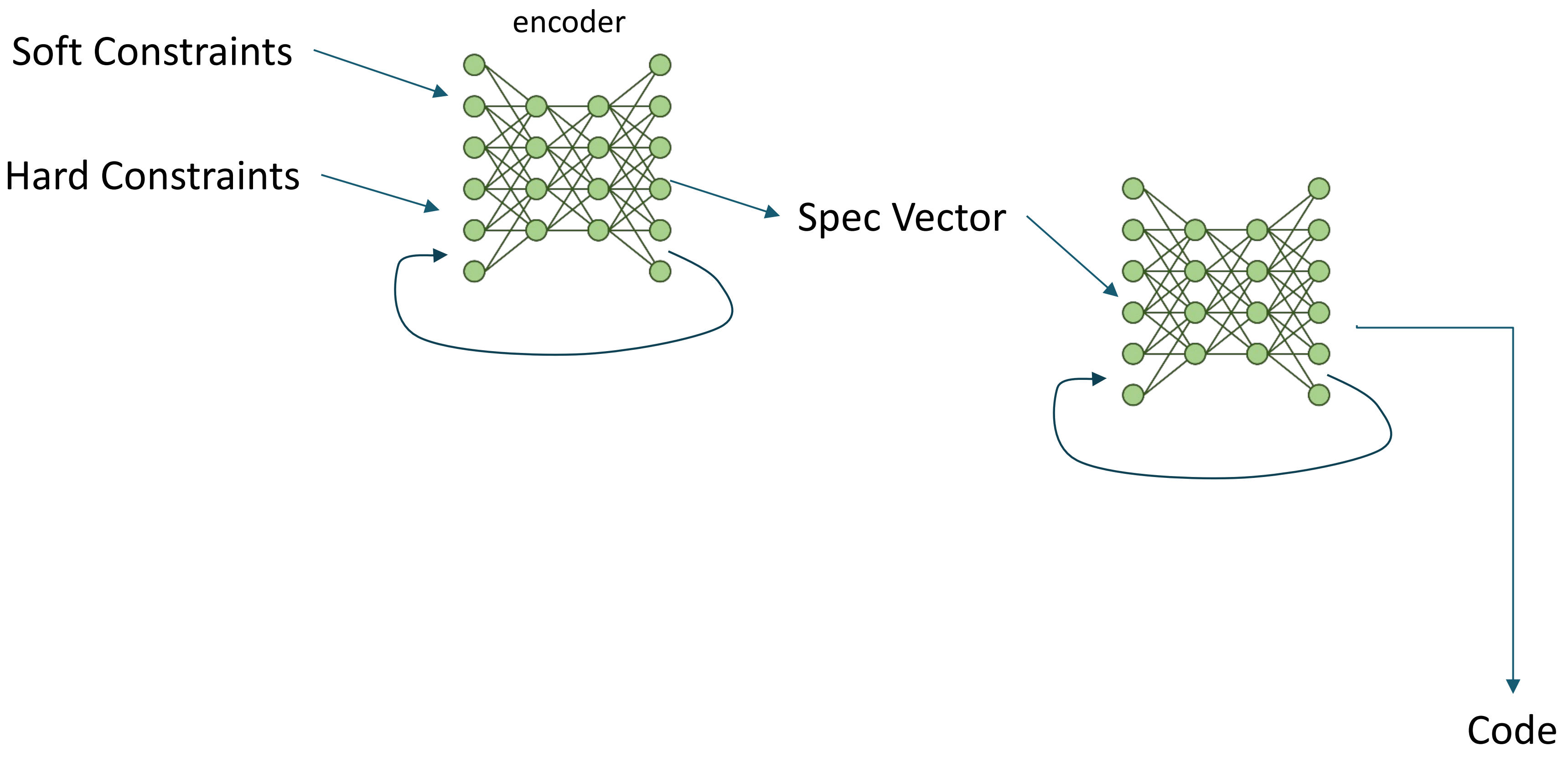
Input

```
void read(String file) {
   /// call:readline
   /// type:BufferedReader

}
```

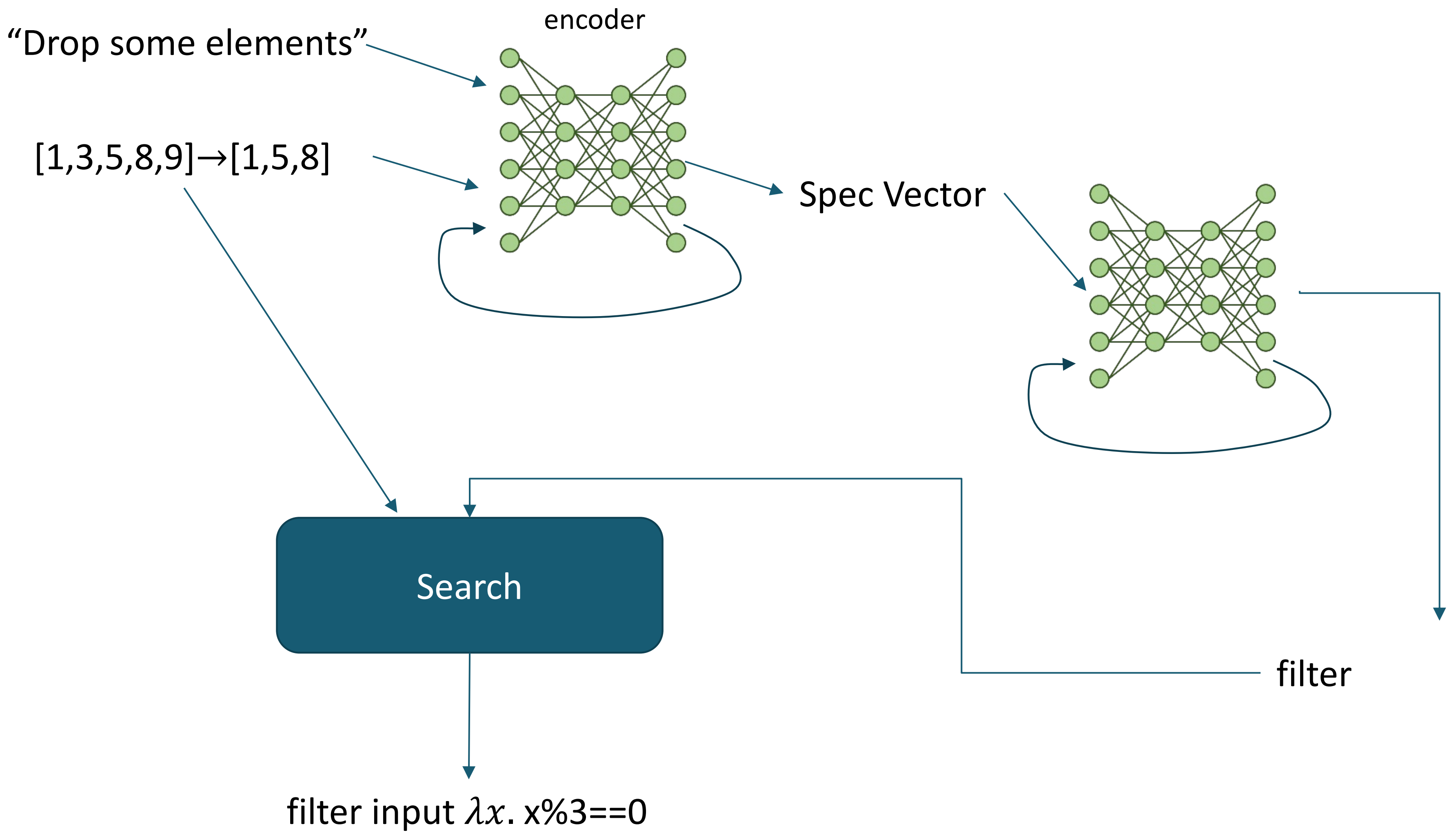
Output

```
void read(String file) {

   String s;
   BufferedReader br;
   FileReader fr;
    try {
       fr = new FileReader(file);

       br = new BufferedReader(fr);

       while ((s = br.readLine())!=null){
        }
       br.close();
    }
   catch (FileNotFoundException _e) {}
   catch (IOException _e) {}
}
```

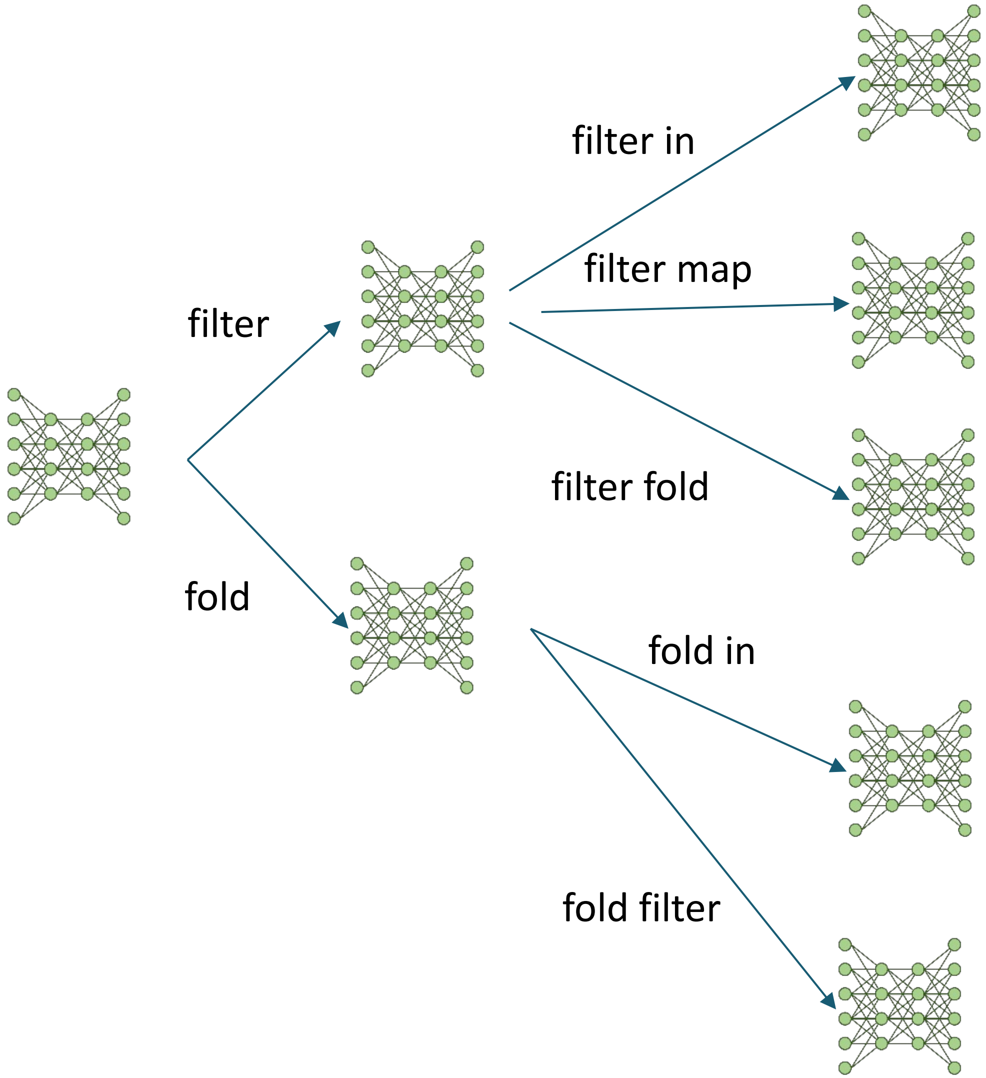
Learning to Sketch



Learning to Sketch



Beam Search + Synthesis



Training

Key idea

Penalize heavily for incorrect token

Penalty for hole depends on cost of completing it

"Drop some elements"

[1,3,5,8,9]→[1,5,8]

filter input λx. x%3==0

fold input λx. x%3==0 ✗

filter input λx. HOLE

HOLE

Algolisp Results