National Key Laboratory for Novel Software Technology

Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code

Hui-Hui Wei and Ming Li

Supervised

Deep Features

Lexical Information

Syntactical Information

Software Functional Clone

Software Clone

Software Clone

- developers reuse code
- a developer implements a functionality that are very similar to an existing one
- software defects, infringement of copyright

Software Clone

- Type-1: identical code fragments in addition to variations in comments and layout;
- Type-2: apart from Type-1 clones, identical code fragments except for different identifier names and literal values; lexicon-based
- Type-3: apart from Type-1 and -2 clones, syntactically similar code that differ at the statement level. The code fragments have statements added, modified and/or removed with respect to each other; syntax-based
- Type-4: syntactically dissimilar code fragments that implement the same functionality. Software Functional Clones

Software Functional Clones

summation implemented with for-loop

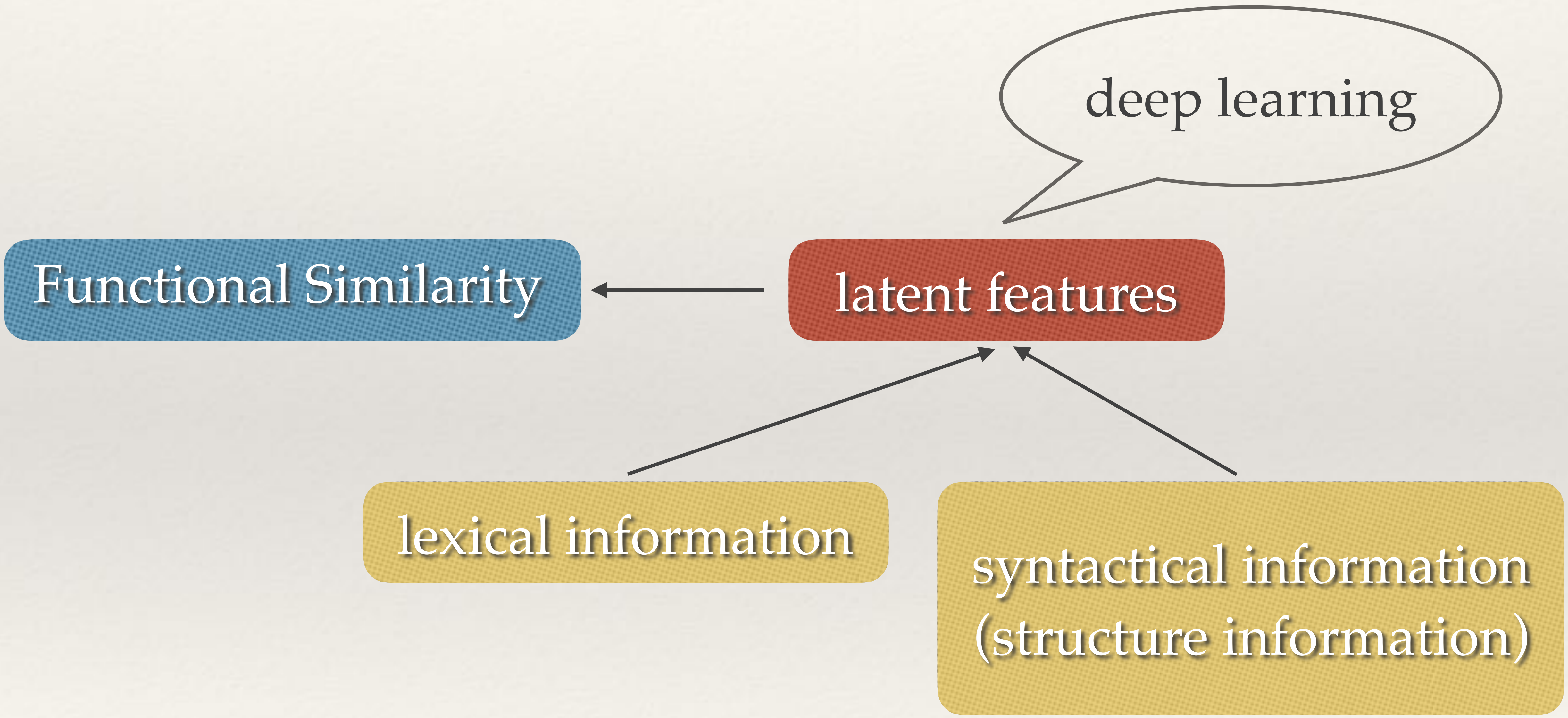
```java
[java]
public class Test {
    private int sum=0,num=1;
    public int calSum(int maxnum){
        if(num<=maxnum){
            sum+=num;
            num++;
            calSum(maxnum);
        }
        return sum;
    }
    public static void main(String[] args) {
        Test test=new Test();
        System.out.println("1+2+3+...+100="+test.calSum(100));
    }
}
```

summation implemented with recursion

```java
int sum = 0;
for (int i = 1; i <= 100; i++) {
sum += i;
}
System.out.println("1到100累加的和为： " + sum);
```

It is difficult to measure the functional similarity simply based on the appearance of the code fragments.

Software Clone



Problem Definition

- n code fragments: $\{C_1, \cdots, C_n\}$
  if (C_i, C_j) is a clone pair, $y_{i,j} = 1$
  if (C_i, C_j) is not a clone pair, $y_{i,j} = -1$
  if (C_i, C_j) is undefined, $y_{i,j} = 0$
- the training set is represented by a set of triplets:
  $D = \{(C_i, C_j, y_{i,j}) \mid i, j \in [n], i<j\}$, $[n]=\{1,2,\cdots,n\}$
- function Φ: maps any pairs of code fragments to $\{-1, 1\}$.

Problem Definition

representation layer:

non-linear representation mapping function ϕ:

$\mathbf{z}_i = \phi(C_i), \forall i \in [n]$

code fragments $\{C_i\}^n \rightarrow$ d-dimensional representations $\{\mathbf{z}_i\}^n$

hashing layer:

a hash function ψ: $R^d \rightarrow \{-1, 1\}^m$

d-dimensional representation $\rightarrow$ m-dimensional Hamming space

$\psi(\mathbf{z}_i) = [h_1(\mathbf{z}_i), h_2(\mathbf{z}_i), \ldots, h_m(\mathbf{z}_i)], \forall i \in [n]$

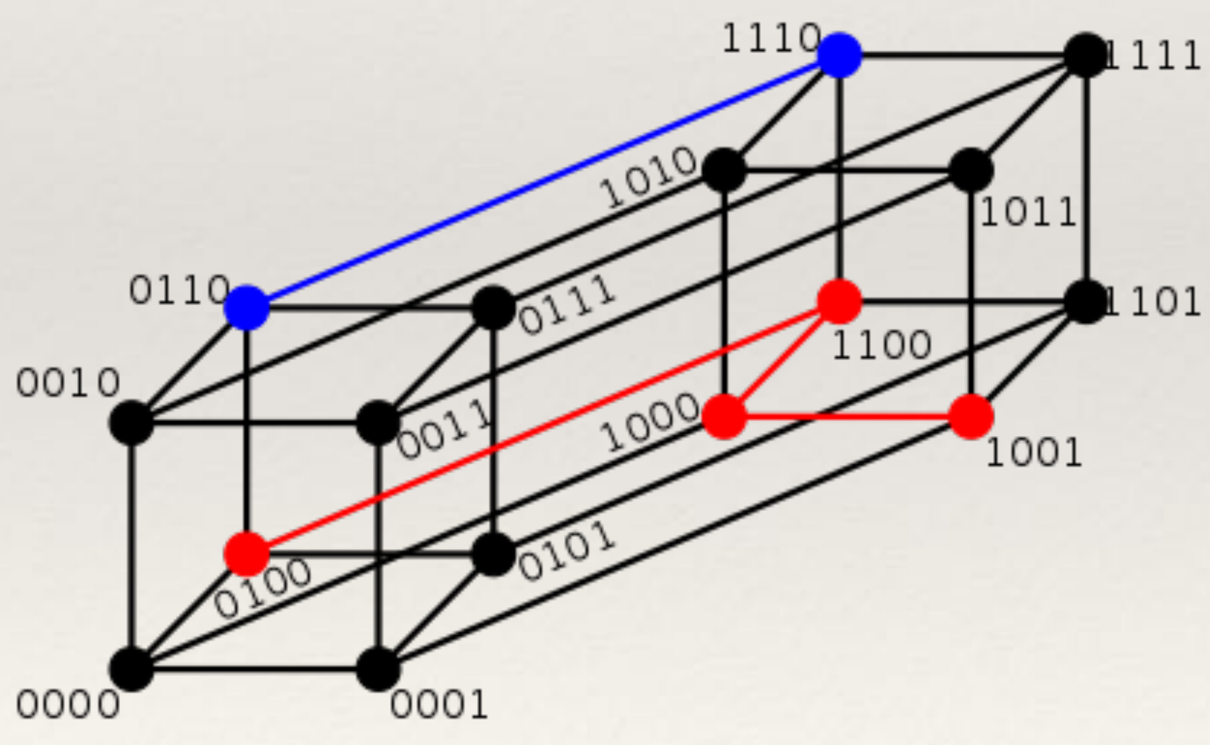
d dimensional representations $\{\mathbf{z}_i\}^n \rightarrow$ binary hash codes $\{\mathbf{a}_i\}^n$

Code fragments belonging to a clone pair can be close to each other in terms of hamming distance, otherwise they should be far away.

Problem Definition

Hamming Distance / Hamming Weight

0100→1001 (red): 3
0110→1110 (blue): 1

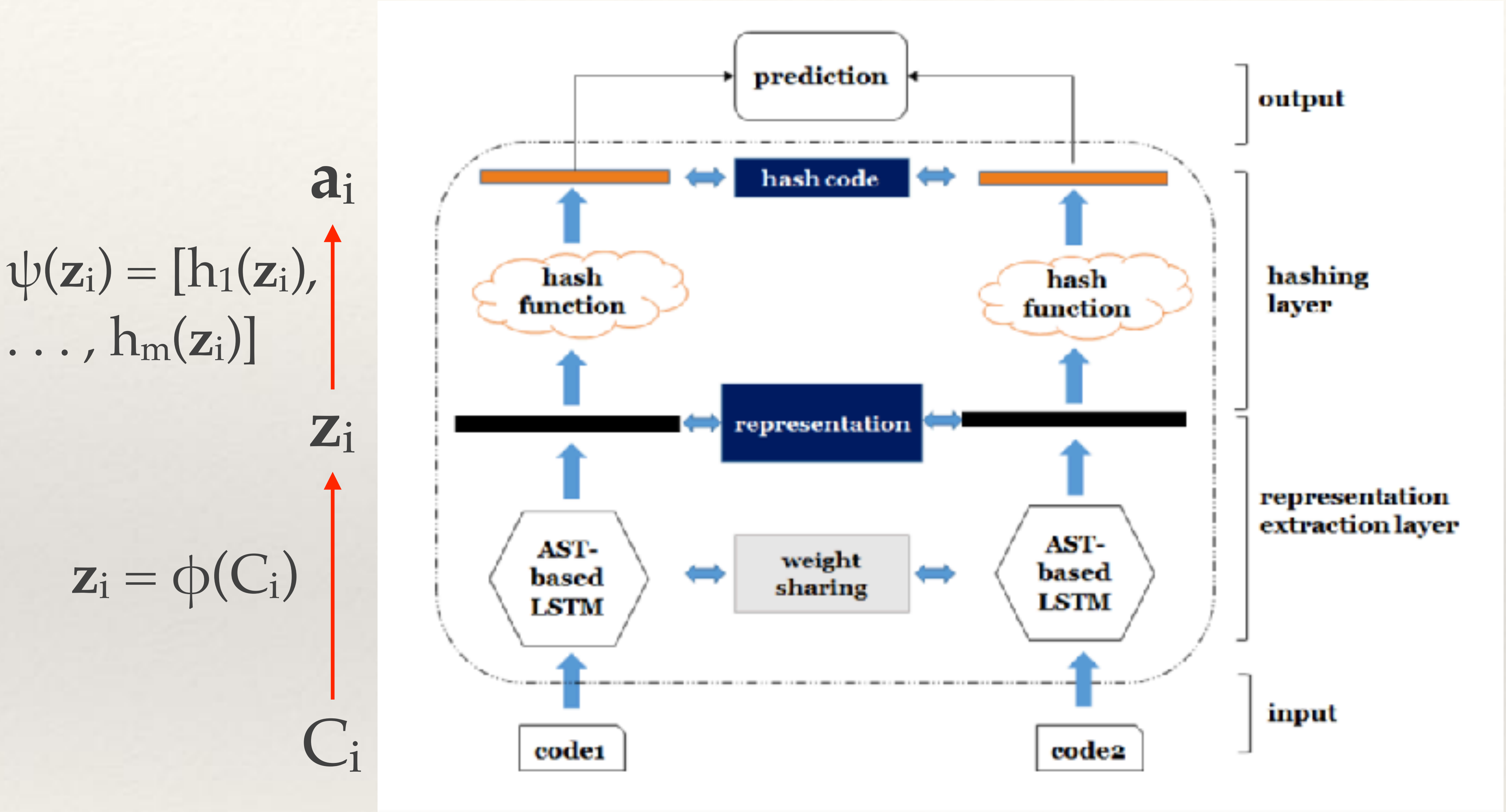


Problem Definition

- code fragments $C_i \rightarrow$ binary hash code $\mathbf{a}_i$
  code fragments $C_j \rightarrow$ binary hash codes $\mathbf{a}_j$
- a pair of ($\mathbf{a}_i$, $\mathbf{a}_j$)
  $$g(\boldsymbol{a}_i, \boldsymbol{a}_j) = \mathbb{I}\left(\sum_{k=1}^{m} 1/4 * (a_{i,k} - a_{j,k})^2 \leq thr\right)$$
  indicator function I(.) :
  if the condition is satisfied, return 1
  otherwise, return -1

$$\Phi(\mathrm{C}_i, \mathrm{C}_j) = g(\psi(\phi(\mathrm{C}_i)), \psi(\phi(\mathrm{C}_j)))$$

CDLH: General Framework



- learn the representation mapping ϕ and the hash function ψ simultaneously.
- force the hash codes for clone pairs to be close to each other, and those for none clone pairs to be far away.

CDLH: General Framework

we define the optimization problem as follows:

$$\min_{\boxed{W,\phi}} \sum_{i=1}^{n}\sum_{j=1}^{n} |y_{i,j}| \left[y_{i,j} - \frac{1}{m}\sum_{k=1}^{m} h_k\left(\phi(\mathrm{C}_i)\right) h_k\left(\phi(\mathrm{C}_j)\right)\right]^2, \quad (1)$$

where $h_k\left(\phi(\mathrm{C}_i)\right) = \mathrm{sign}(\boldsymbol{w}_k^\top \phi(\mathrm{C}_i) + b_k)$, b_k is a bias term, $W = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_m, b_1, \cdots, b_m\}$.

- representation mapping function ϕ: incorporates both the lexical and syntactical information of source codes

AST-based LSTM

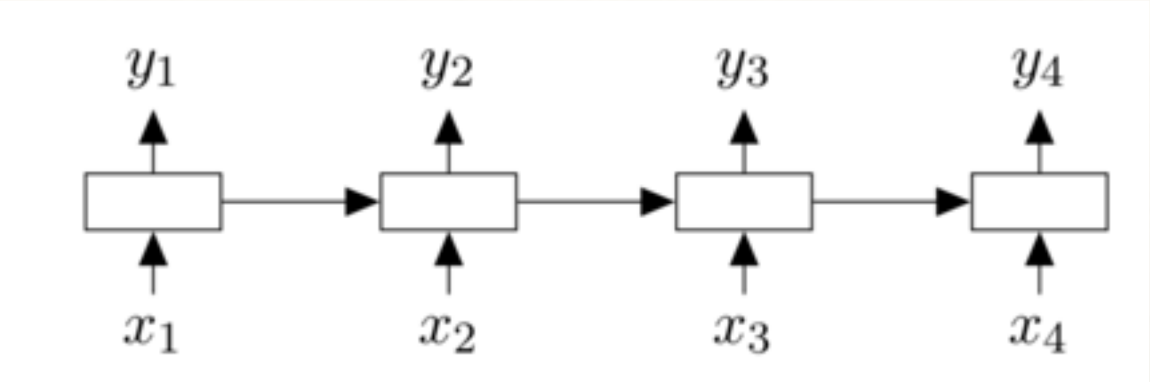
syntactical (structure) information

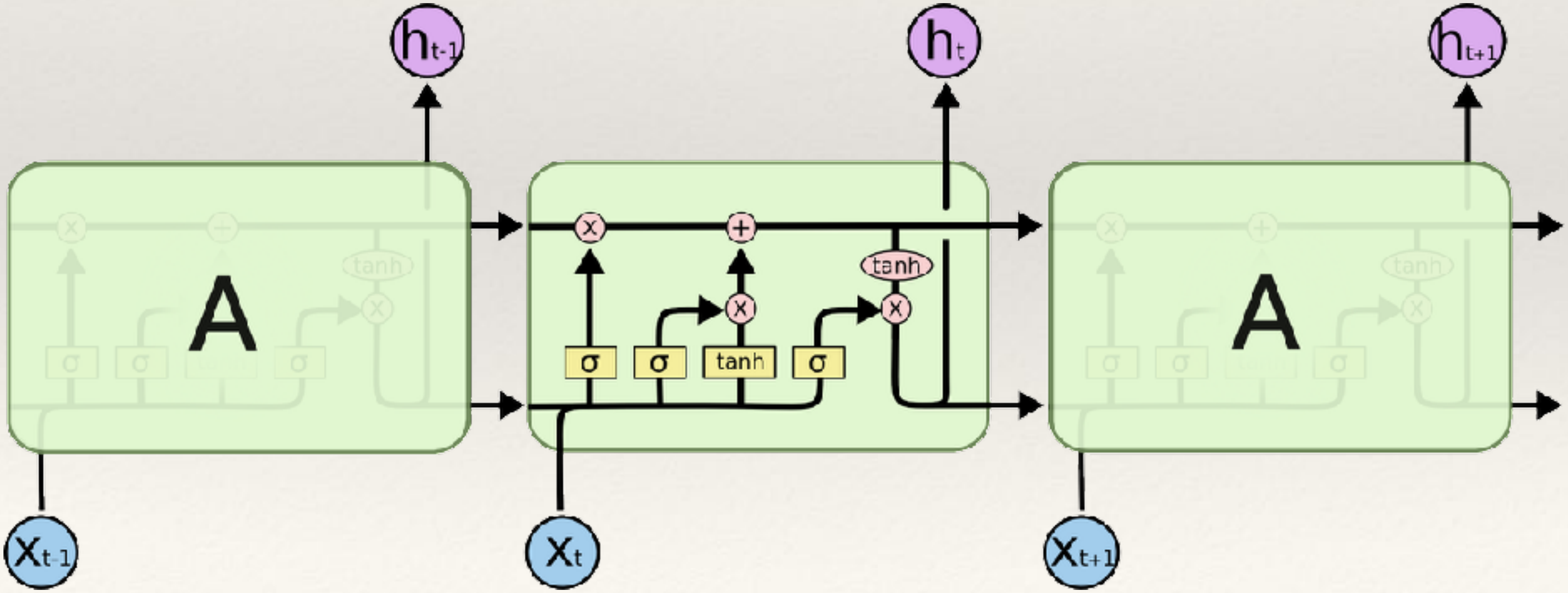
- AST: capture structure information of code fragments.
- LSTM: extract the semantic information carried by lexical tokens of source codes.

lexical information

LSTM

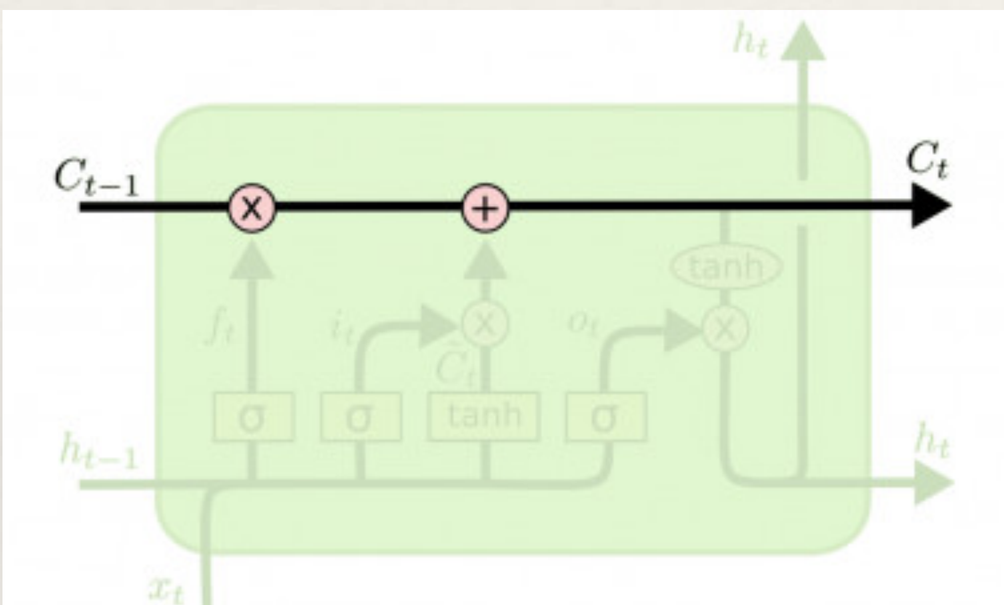
chain-structured LSTM network



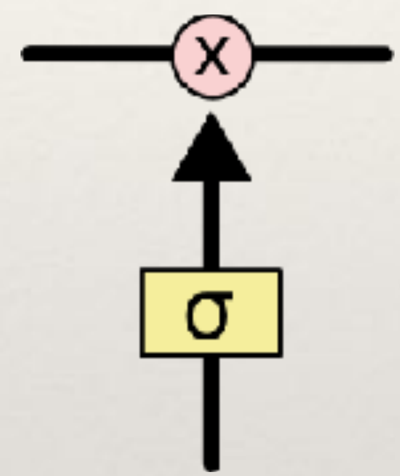


LSTM

memory cell



gate

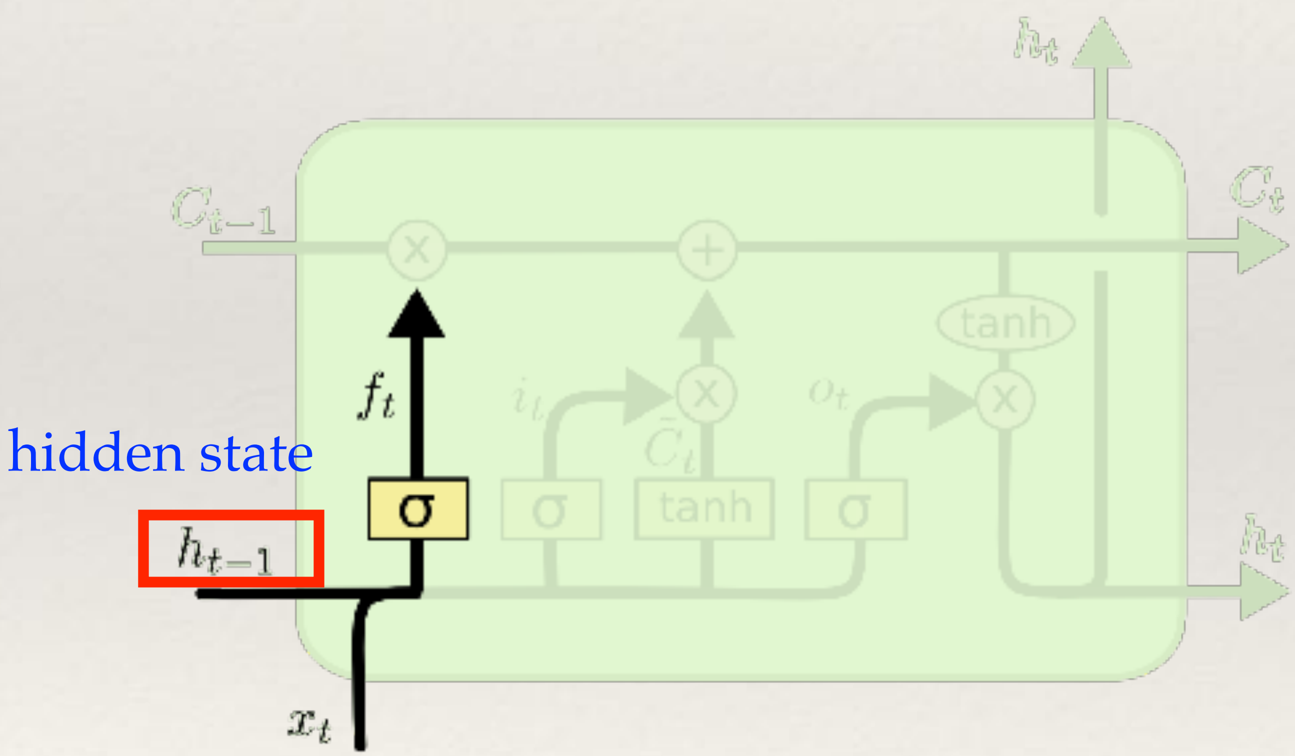


- input gate
- forget gate
- output gate

a sigmoid neural net layer
a pointwise multiplication operation

LSTM

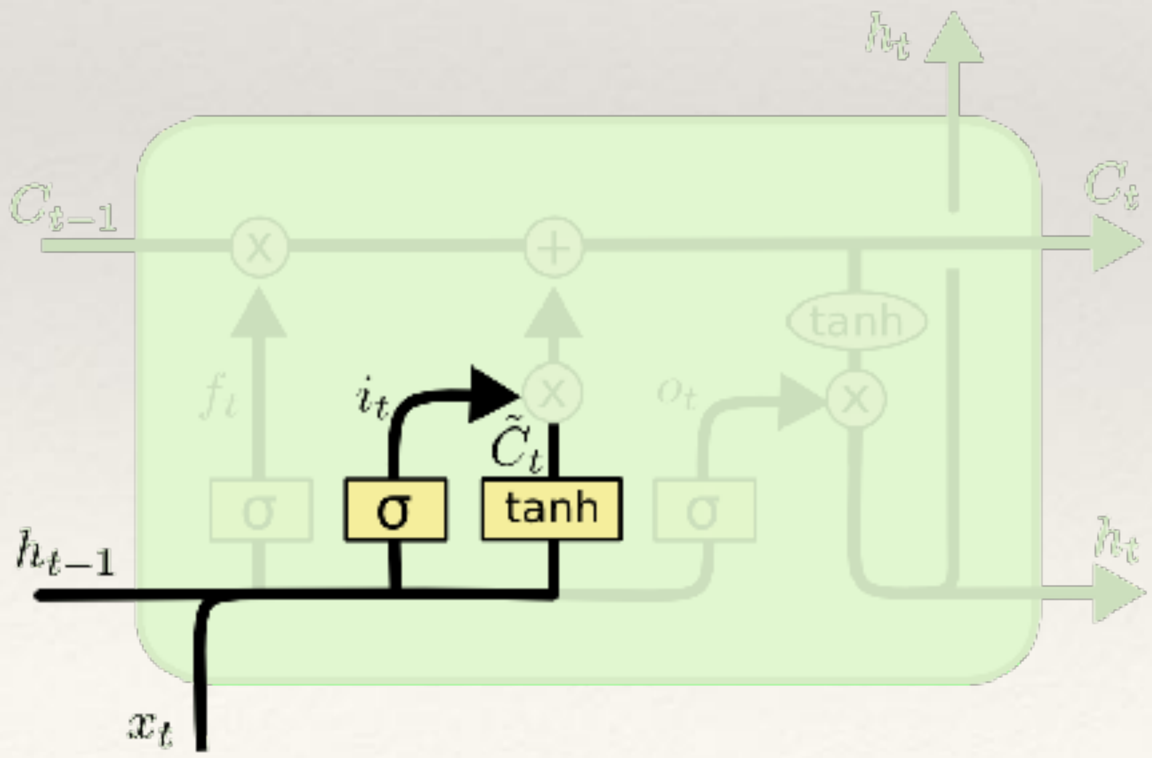
- Step 1: decide what information we're going to throw away from the memory cell. — forget gate layer



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM

- Step 2: decide what new information we're going to store in the cell state.
  - First: input gate layer decides which values we'll update.
  - Second: a tanh layer creates a vector of new candidate values, $\tilde{C_t}$,that could be added to the state.

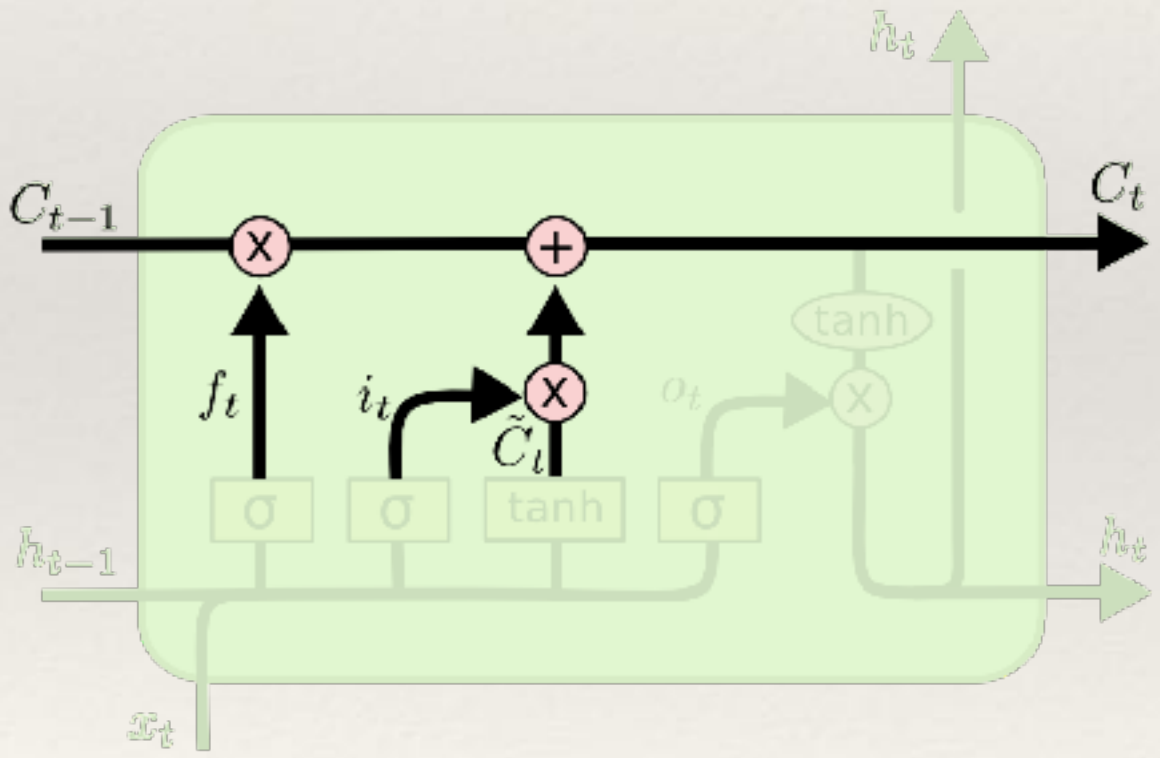


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

LSTM

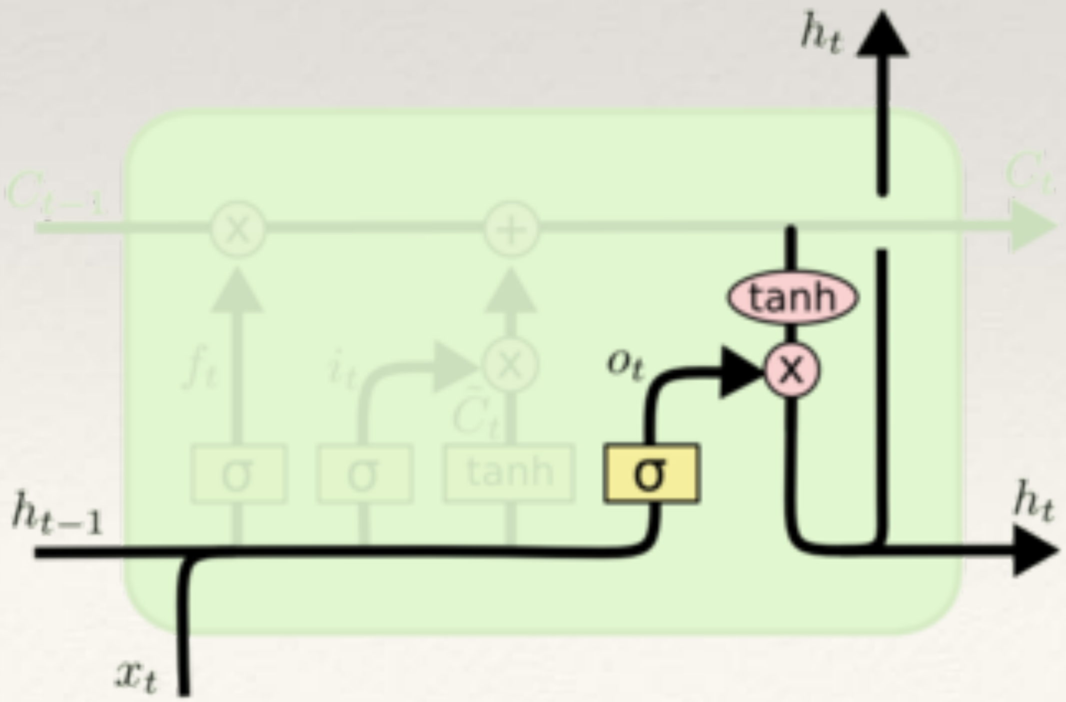
- Step 3: update the old cell state, C_{t-1}, into the new cell state C_t.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM

- Step 4: decide what we're going to output. This output will be based on our cell state, but will be a filtered version.
  - First: we run a sigmoid layer which decides what parts of the cell state we're going to output.
  - Second: we only output the parts we decided to.

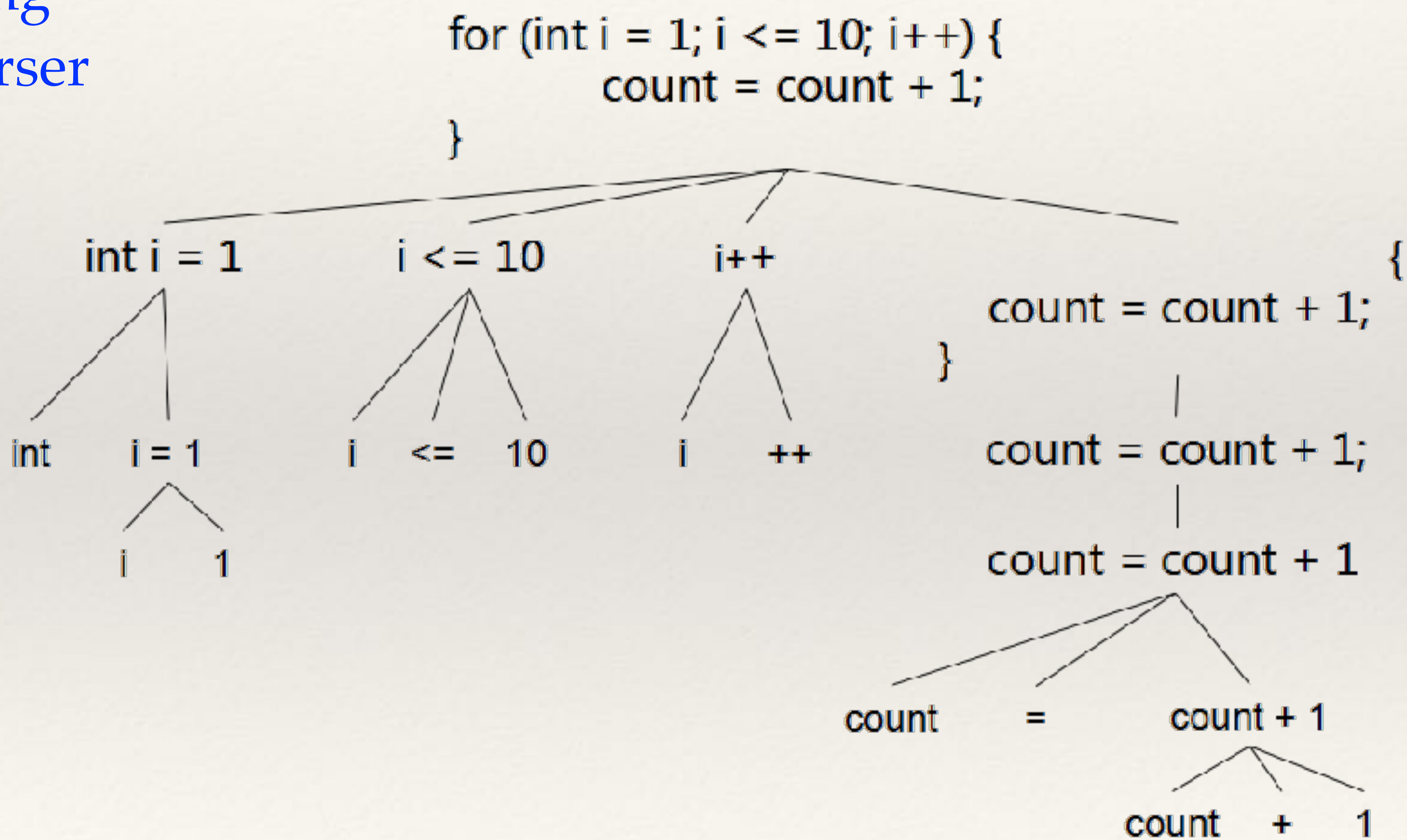


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

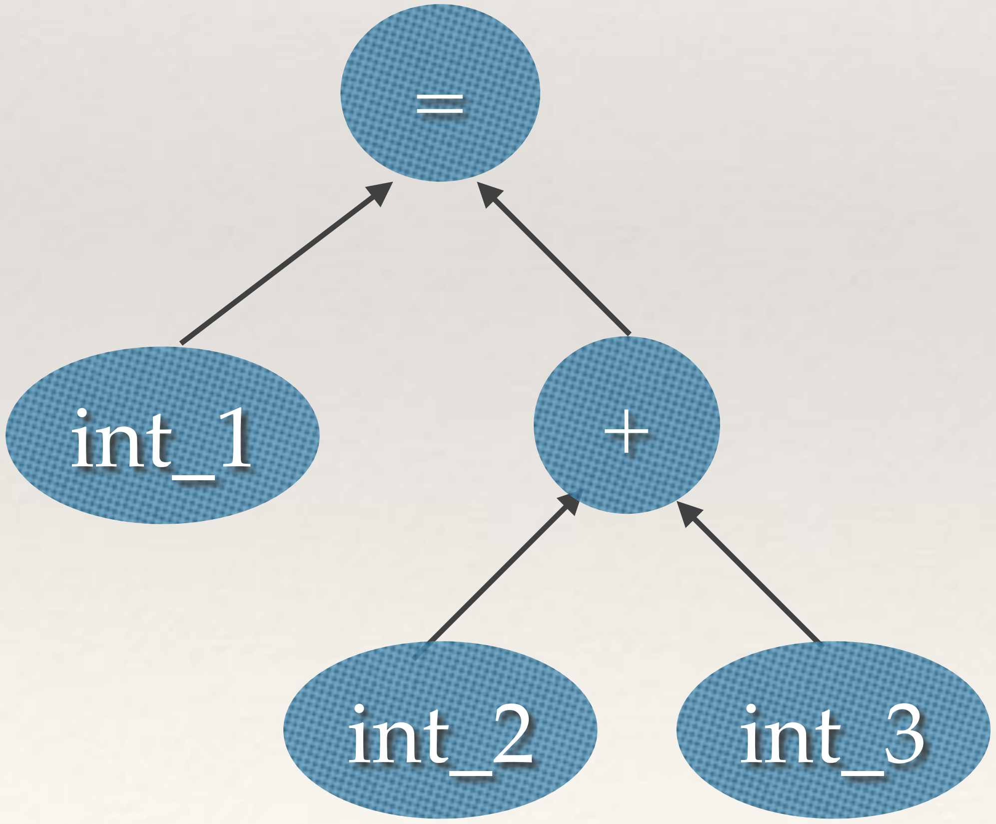
AST: Abstract Syntax Tree

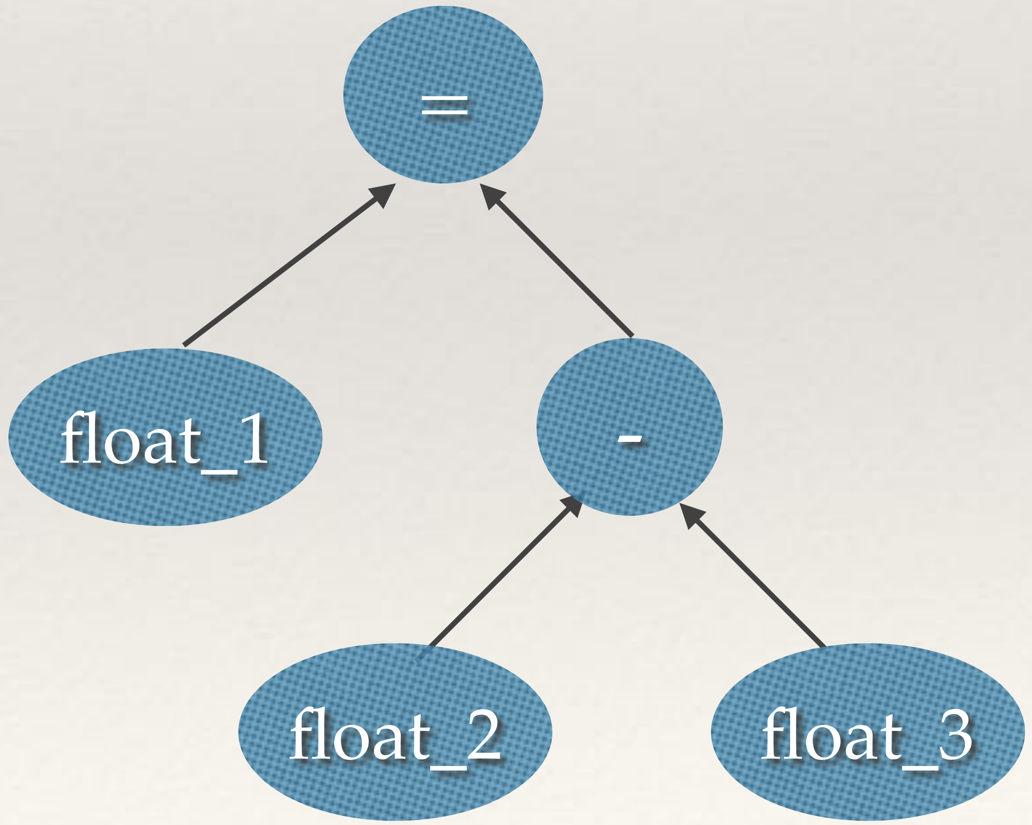
Eclipse JDT → ASTParser
javalang
pycparser



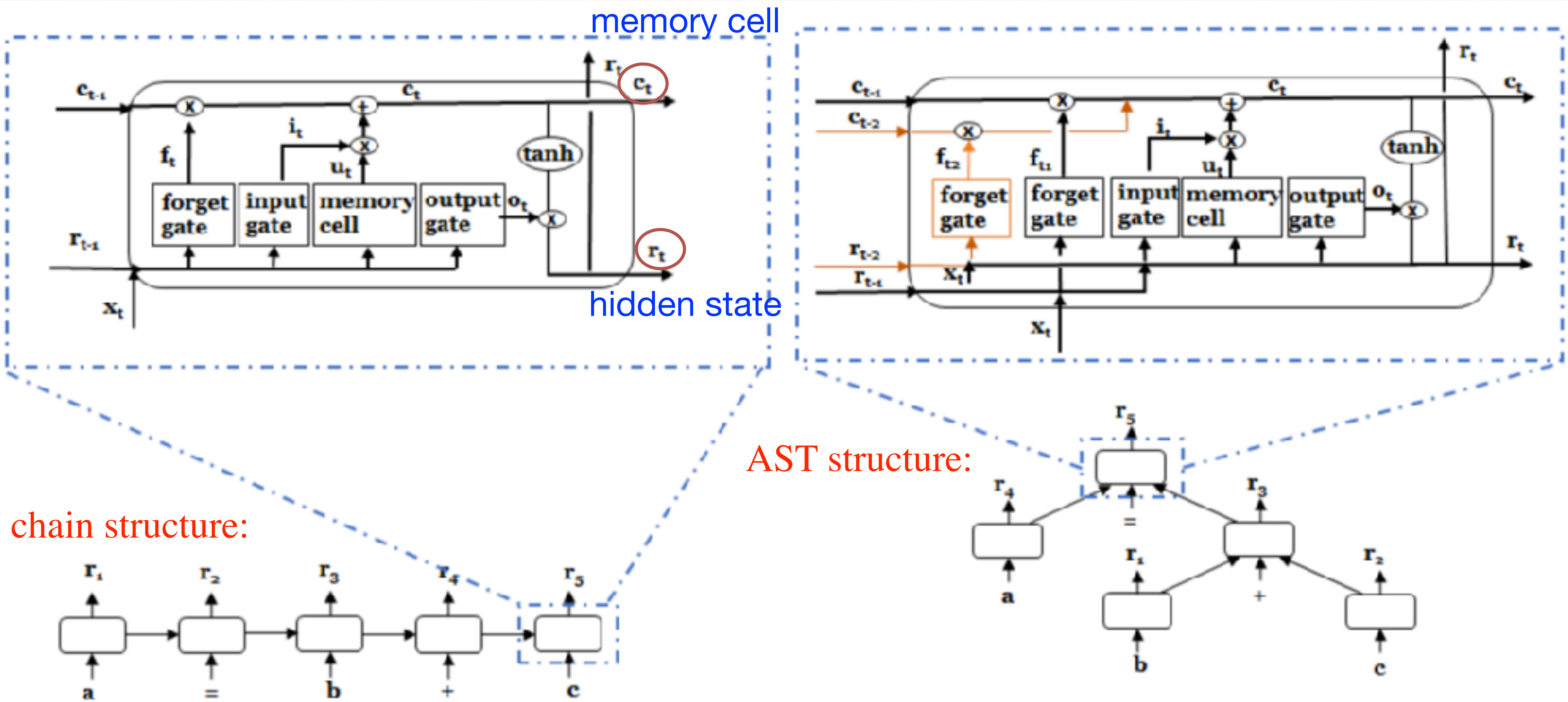
AST: Abstract Syntax Tree

- int_1 = int_2 + int_3;
- float_1 = float_2 – float_3;   / / a simple subtraction operation





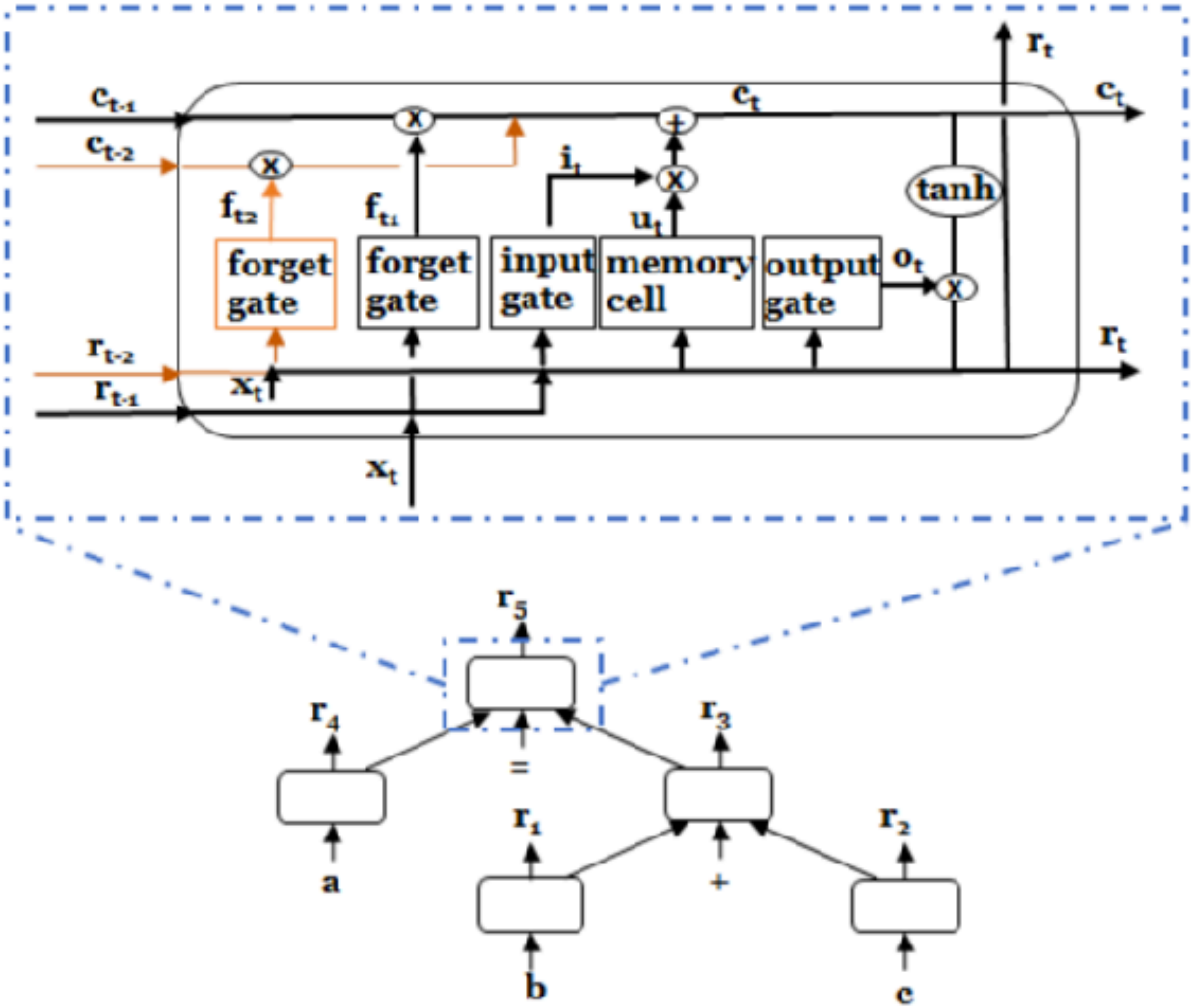
AST-based LSTM



(a) Traditional LSTM for expression 'a=b+c;'

(b) AST-based LSTM for expression 'a=b+c;'

AST-based LSTM



(b) AST-based LSTM for expression 'a=b+c;'

$$\boldsymbol{i} = \sigma(W_i \boldsymbol{x} + \sum_{l=1}^{L} U_{il} \boldsymbol{r}_l + \boldsymbol{b}_i),$$

$$\boldsymbol{f}_l = \sigma(W_f \boldsymbol{x} + U_{fl} \boldsymbol{r}_l + \boldsymbol{b}_f), \; l = 1, 2, \ldots, L$$

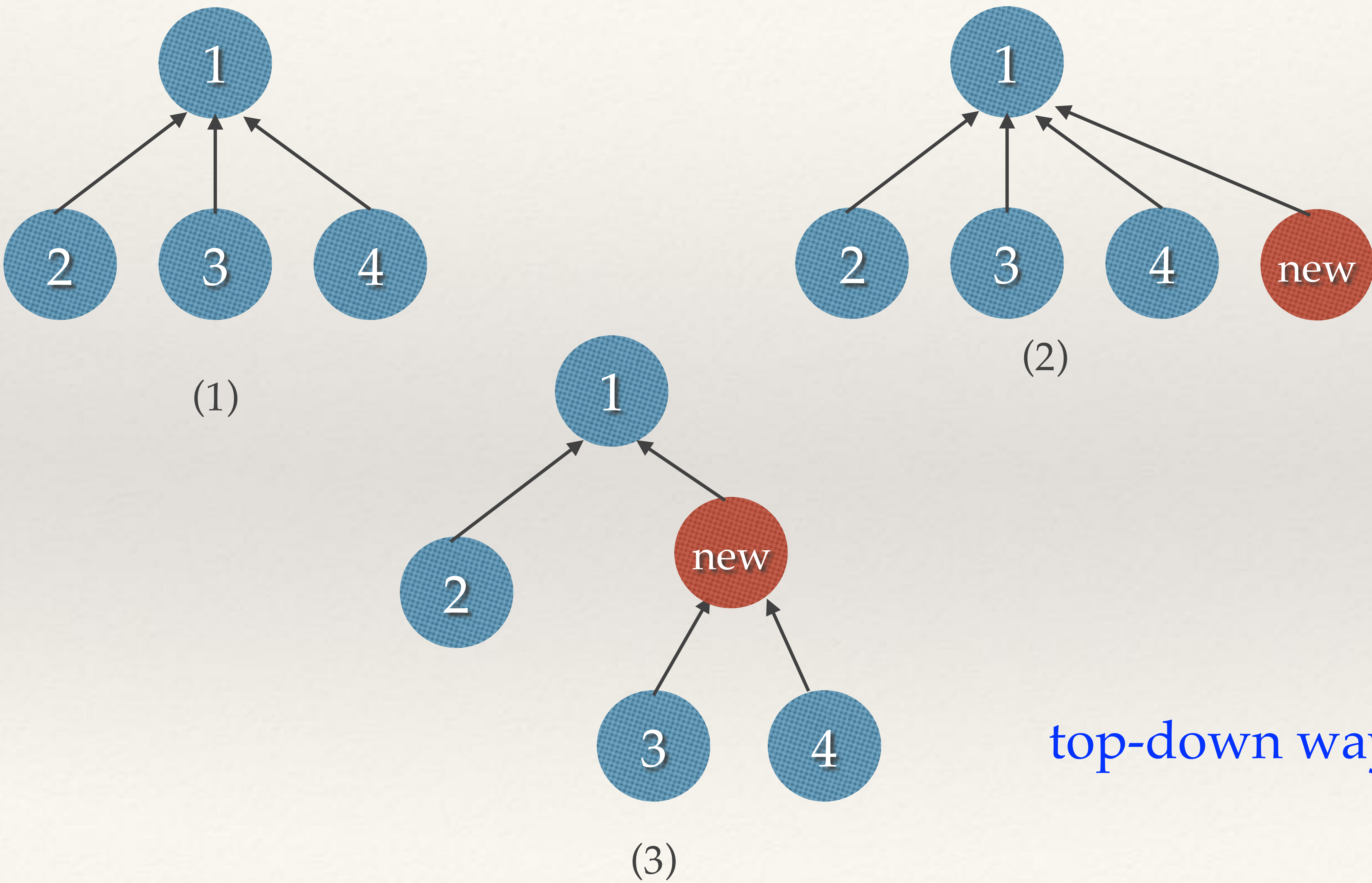
$$\boldsymbol{o} = \sigma(W_o \boldsymbol{x} + \sum_{l=1}^{L} U_{ol} \boldsymbol{r}_l + \boldsymbol{b}_o),$$

$$\boldsymbol{u} = \tanh(W_u \boldsymbol{x} + \sum_{l=1}^{L} U_{ul} \boldsymbol{r}_l + \boldsymbol{b}_u),$$

$$\boldsymbol{c} = \boldsymbol{i} \odot \boldsymbol{u} + \sum_{l=1}^{L} \boldsymbol{f}_l \odot \boldsymbol{c}_l,$$

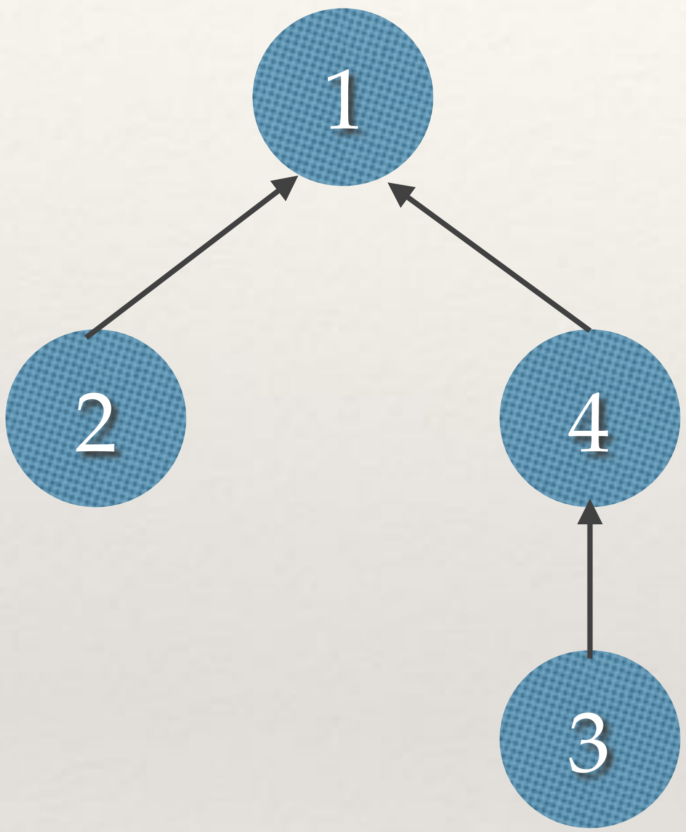
$$\boldsymbol{r} = \boldsymbol{o} \odot \tanh(\boldsymbol{c}),$$

AST-based LSTM

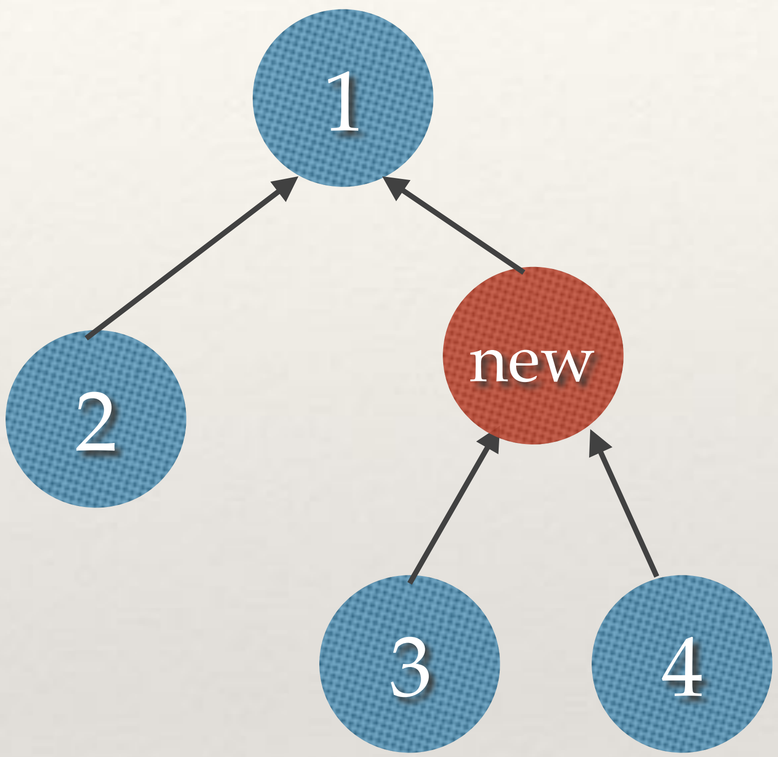


top-down way

AST-based LSTM



(1)



(2)