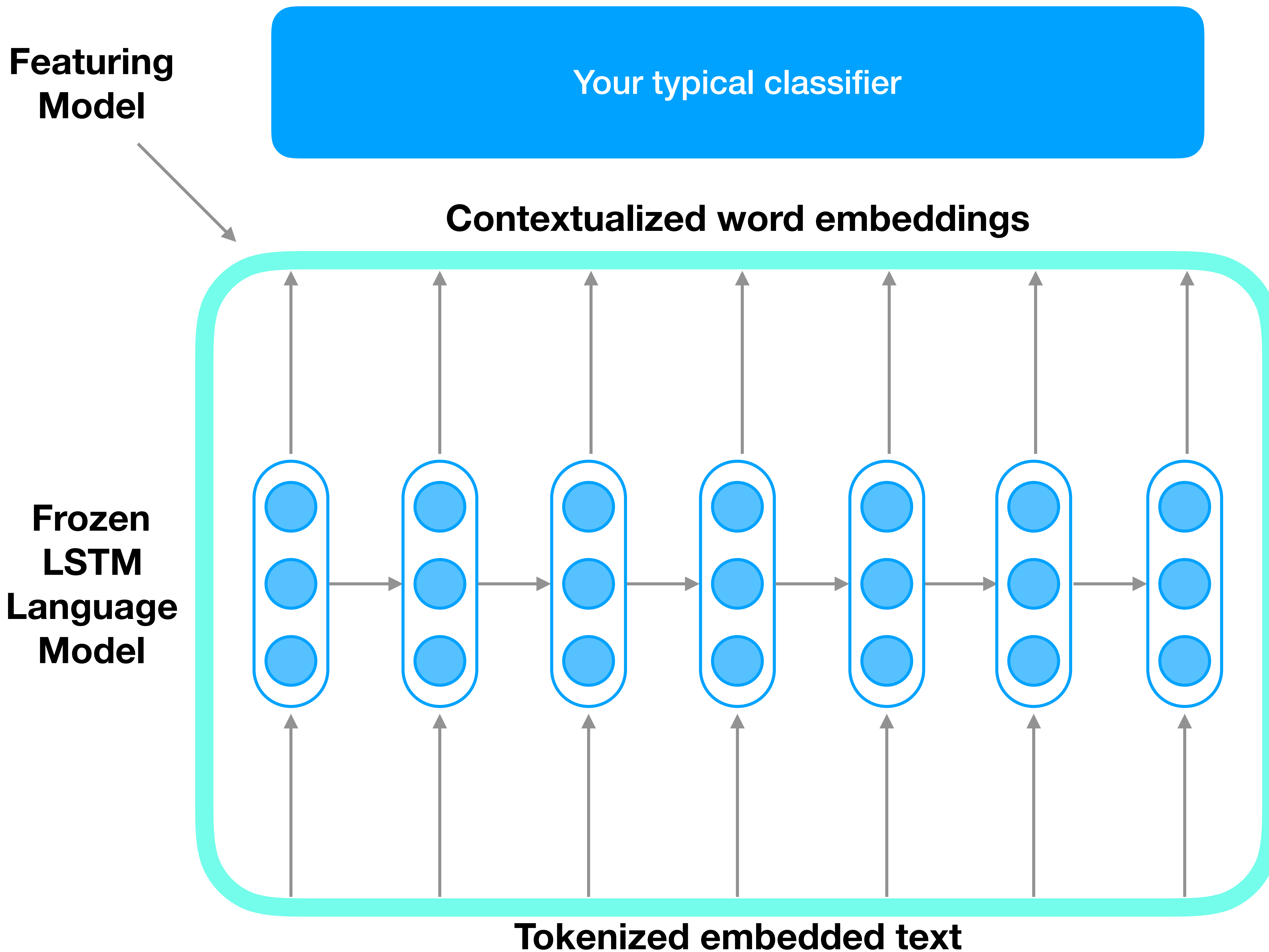
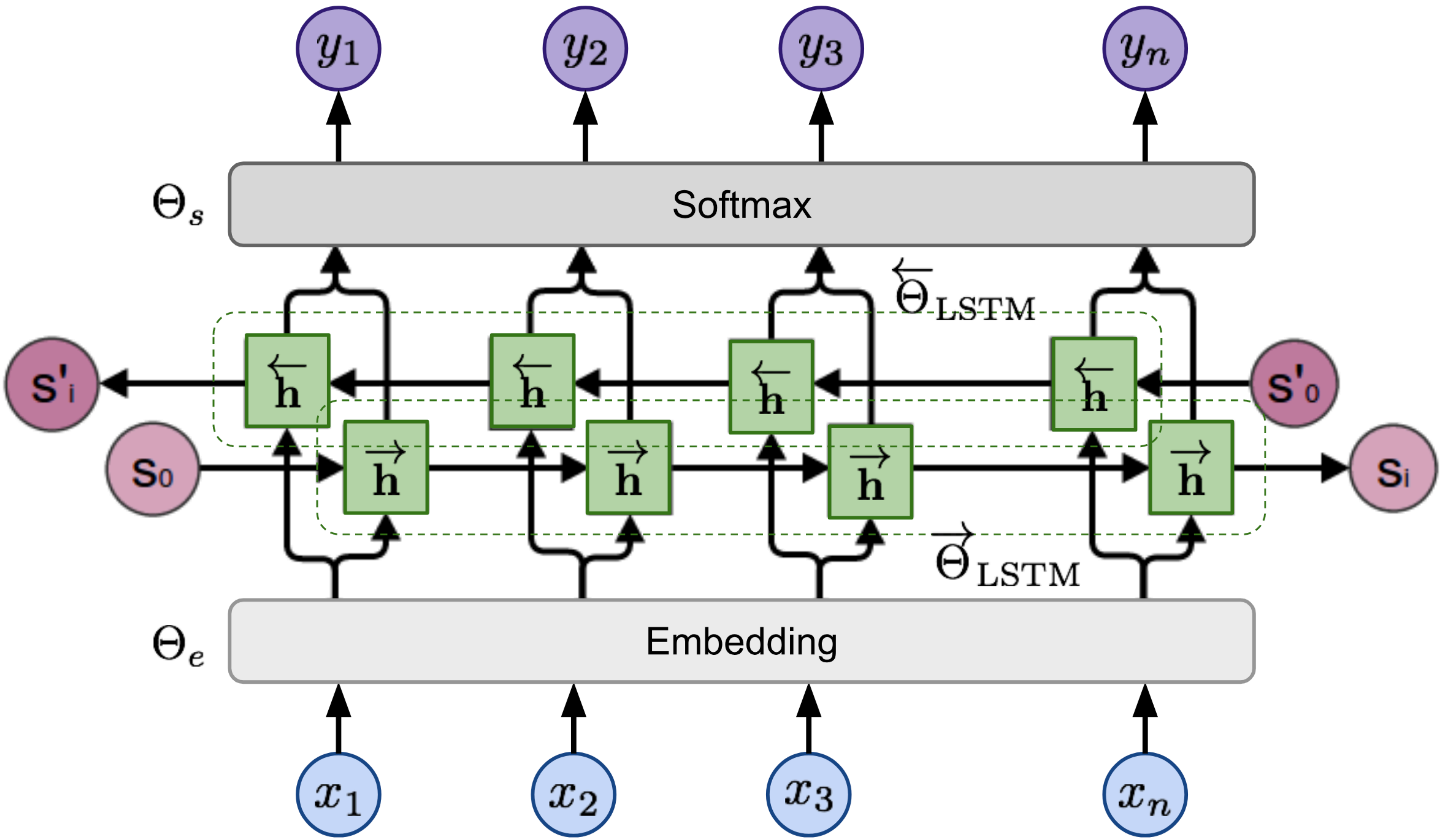
# NLP LM Transfer Learning



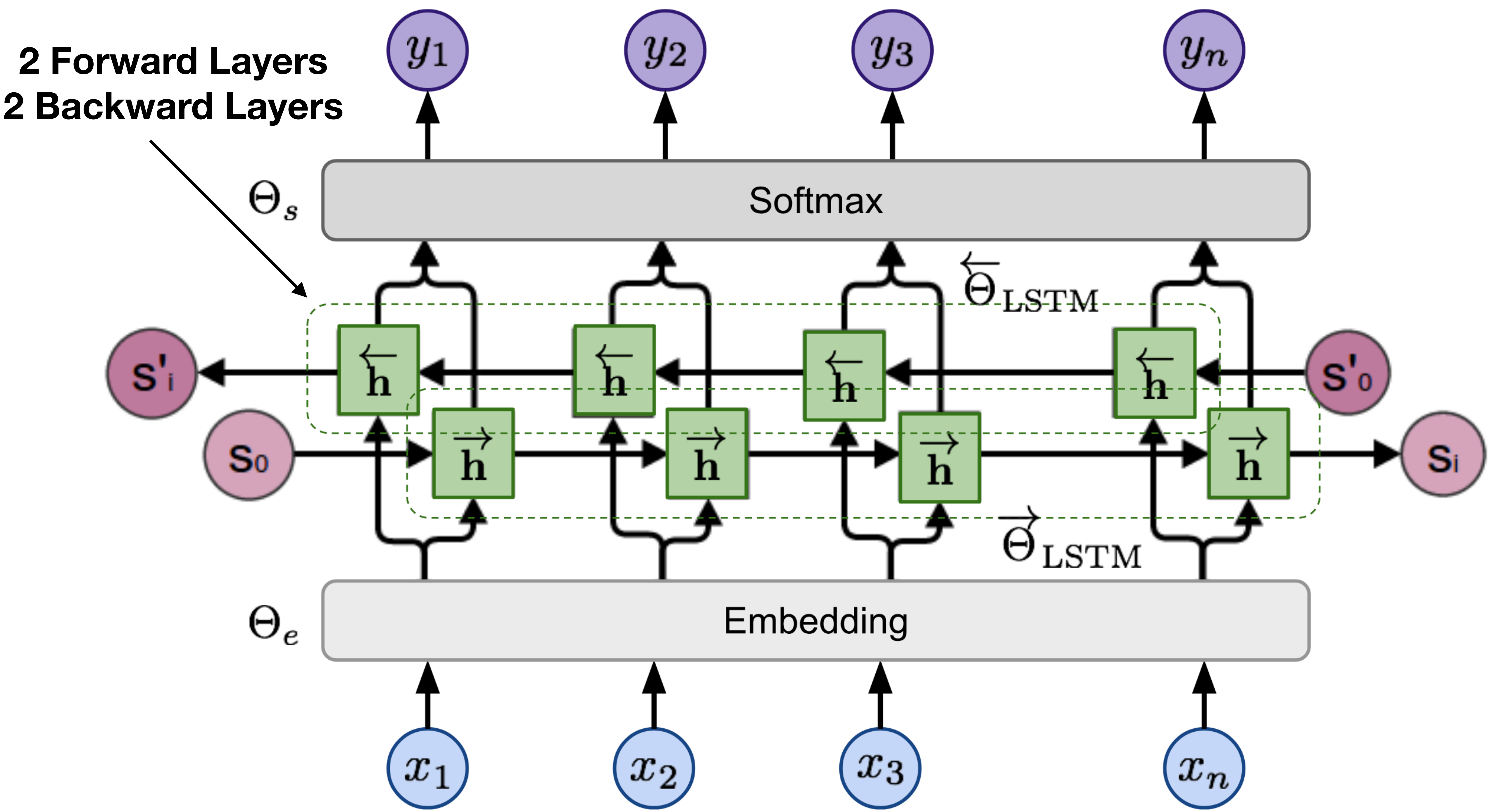
# ELMo

# ELMo

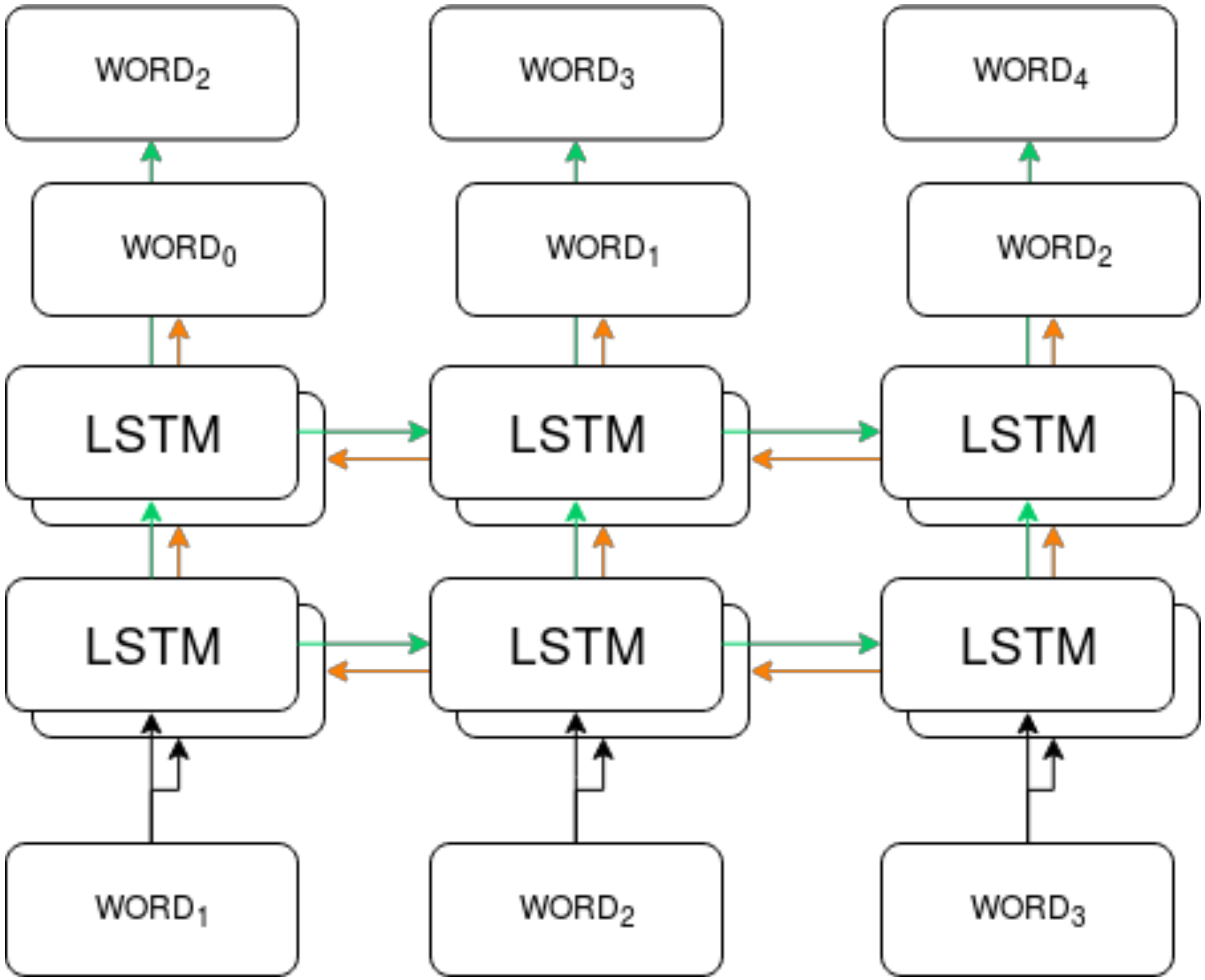


$y_1$
$y_2$
$y_3$
$y_n$
$\Theta_s$
Softmax
$\overleftarrow{\Theta}_{\mathrm{LSTM}}$
s'i
s'0
s0
si
$\overleftarrow{\mathbf{h}}$
$\overrightarrow{\mathbf{h}}$
$\overrightarrow{\Theta}_{\mathrm{LSTM}}$
$\Theta_e$
Embedding
$x_1$
$x_2$
$x_3$
$x_n$

# ELMo


2 Forward Layers
2 Backward Layers
$y_1$
$y_2$
$y_3$
$y_n$
$\Theta_s$
Softmax
$\overleftarrow{\Theta}_{\text{LSTM}}$
S'i
S'0
S0
Si
$\overleftarrow{\mathbf{h}}$
$\overrightarrow{\mathbf{h}}$
$\overrightarrow{\Theta}_{\text{LSTM}}$
$\Theta_e$
Embedding
$x_1$
$x_2$
$x_3$
$x_n$

# ELMo

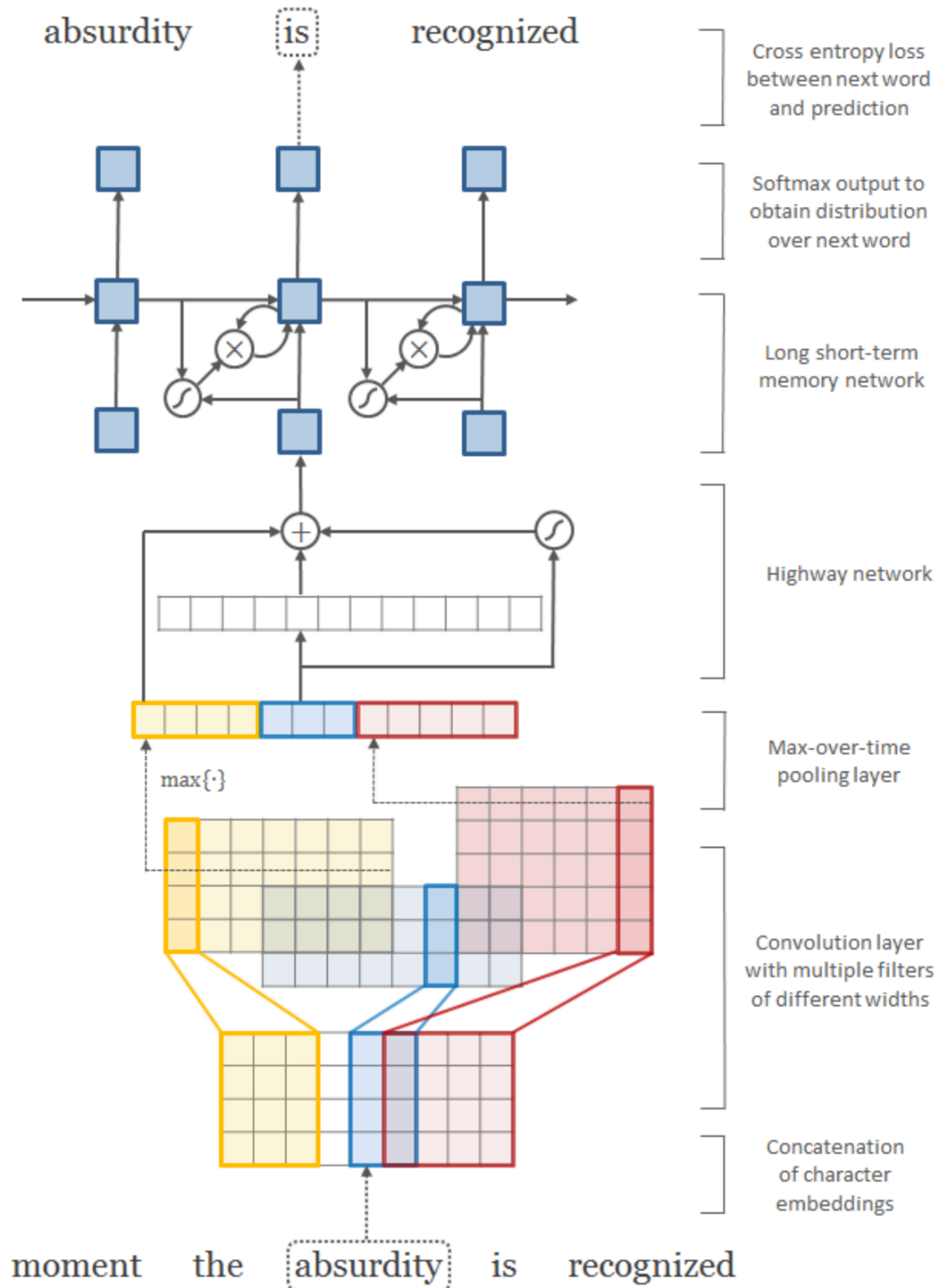


WORD$_2$
WORD$_3$
WORD$_4$
WORD$_0$
WORD$_1$
WORD$_2$
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
WORD$_1$
WORD$_2$
WORD$_3$

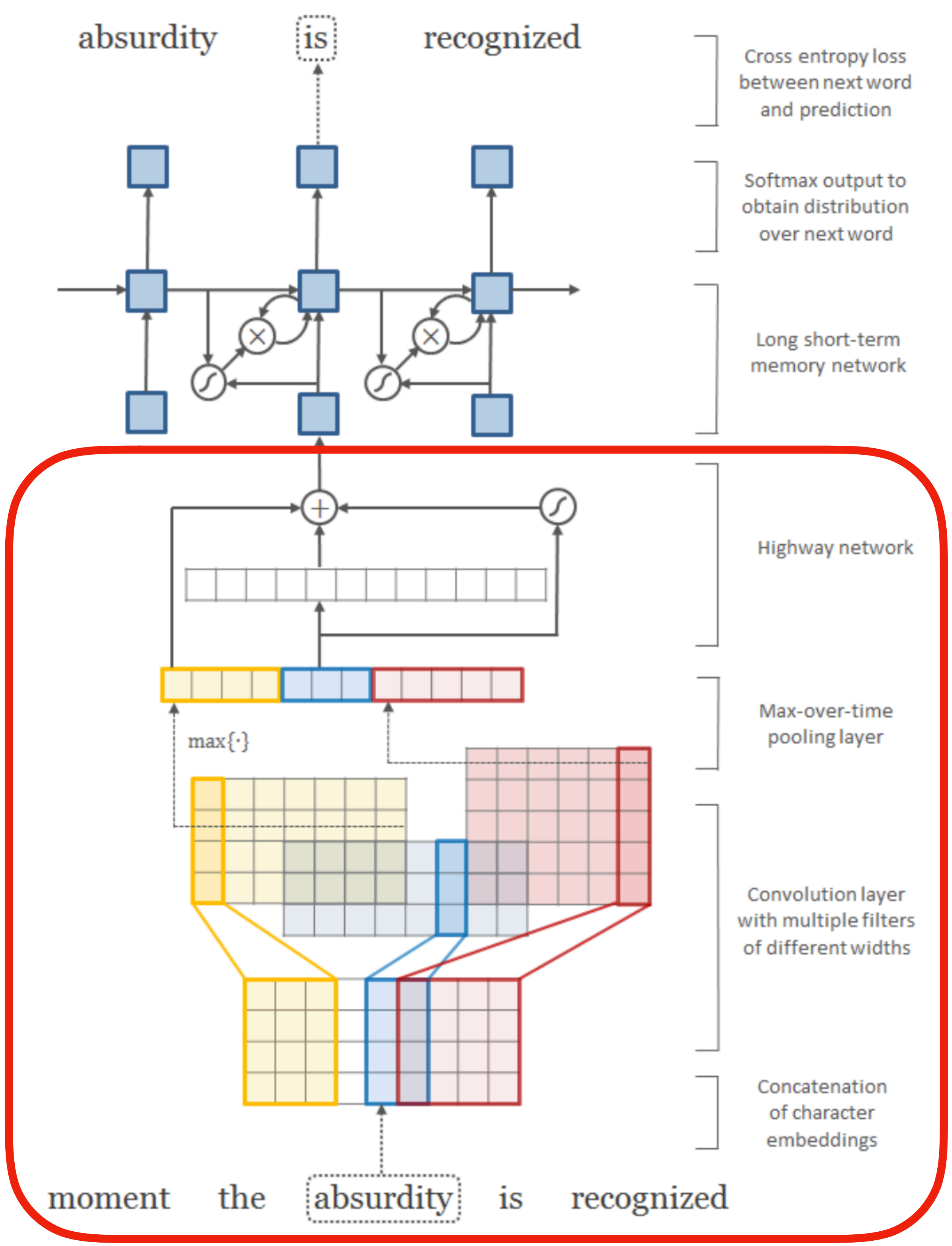
# ELMo

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Source** | <START> | The | poor | don’t | have | any | money | <END> |
| **Forward** | <START> | The | poor | don’t | have | any | money | <END> |
| **Backward** | <END> | money | any | have | don’t | poor | The | <START> |

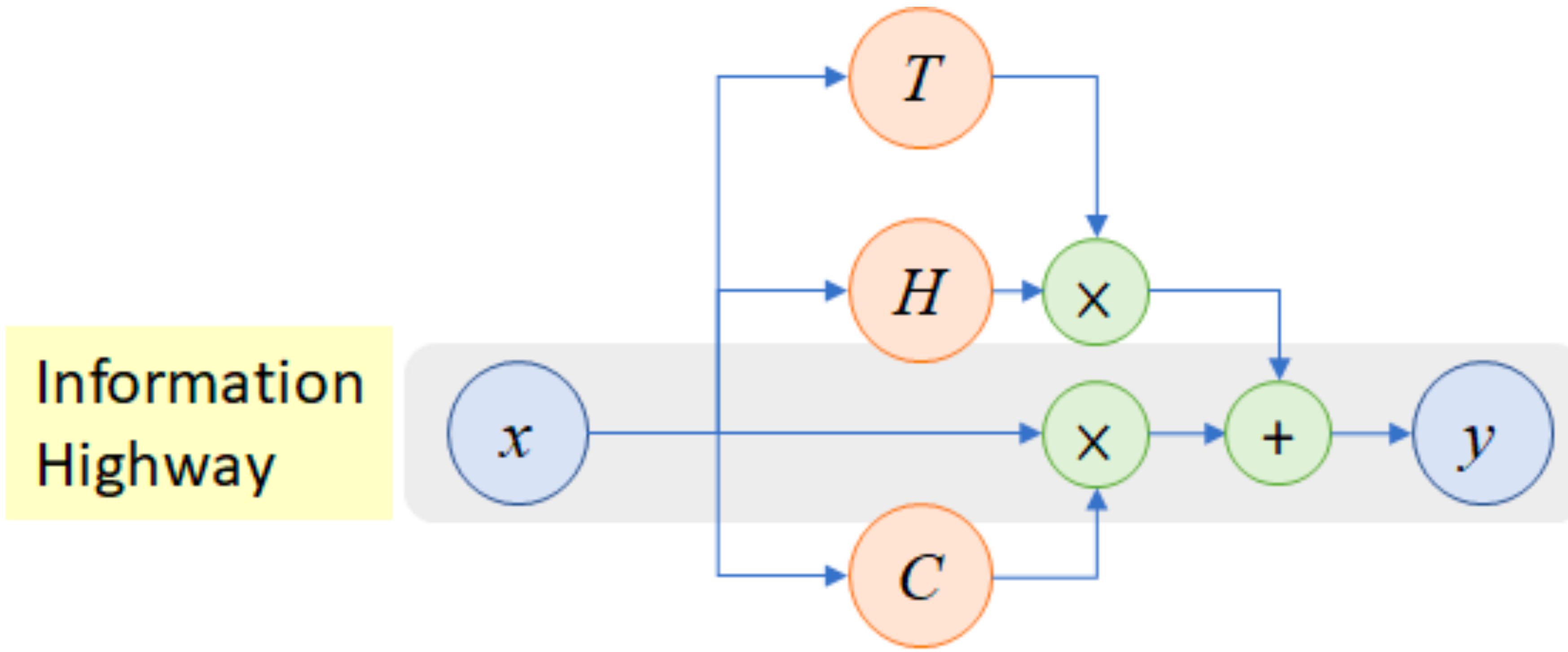
# Character-Aware Neural Language Models



# Character-Aware Neural Language Models



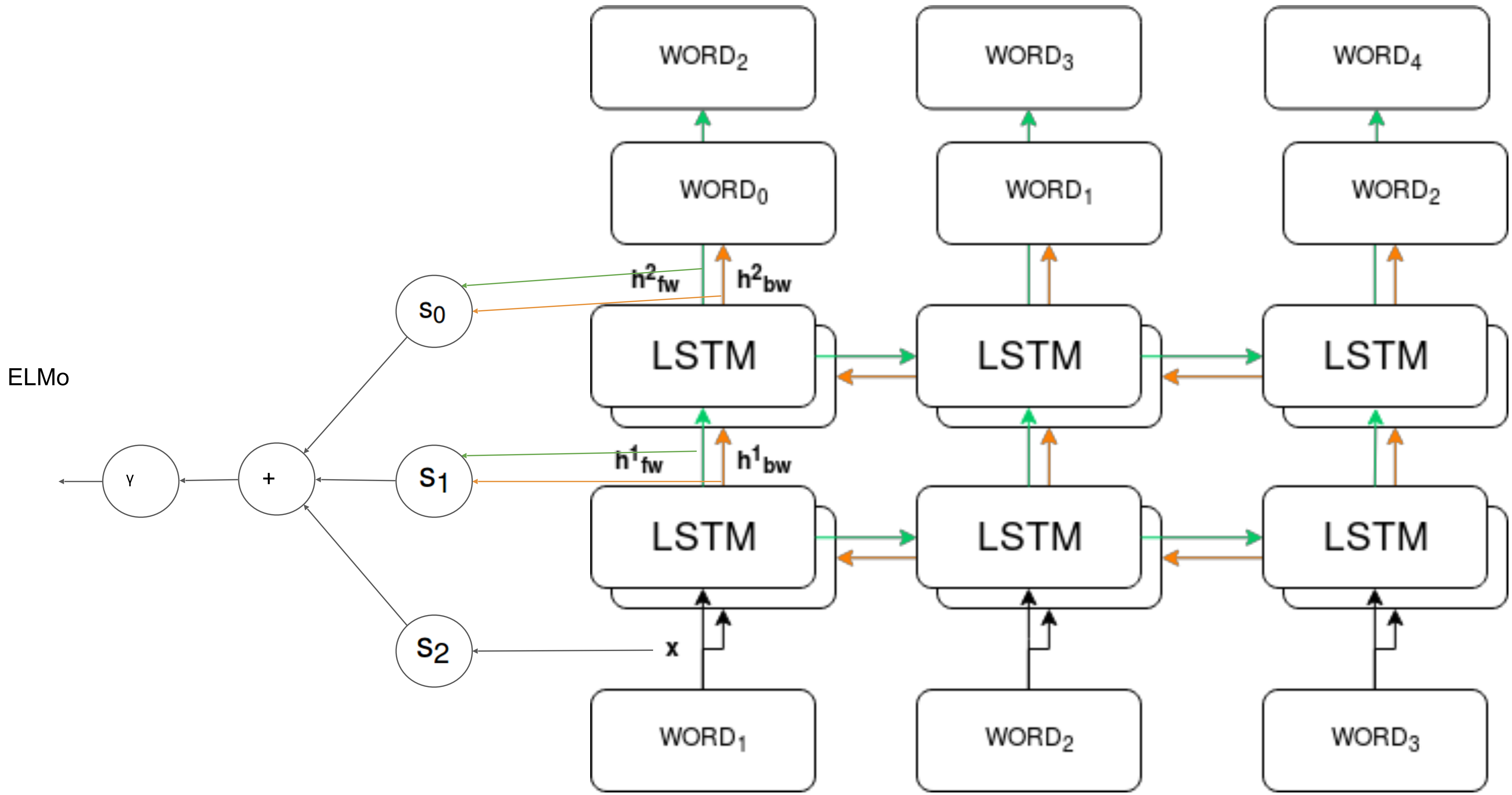
# Highway Network




$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}).$$

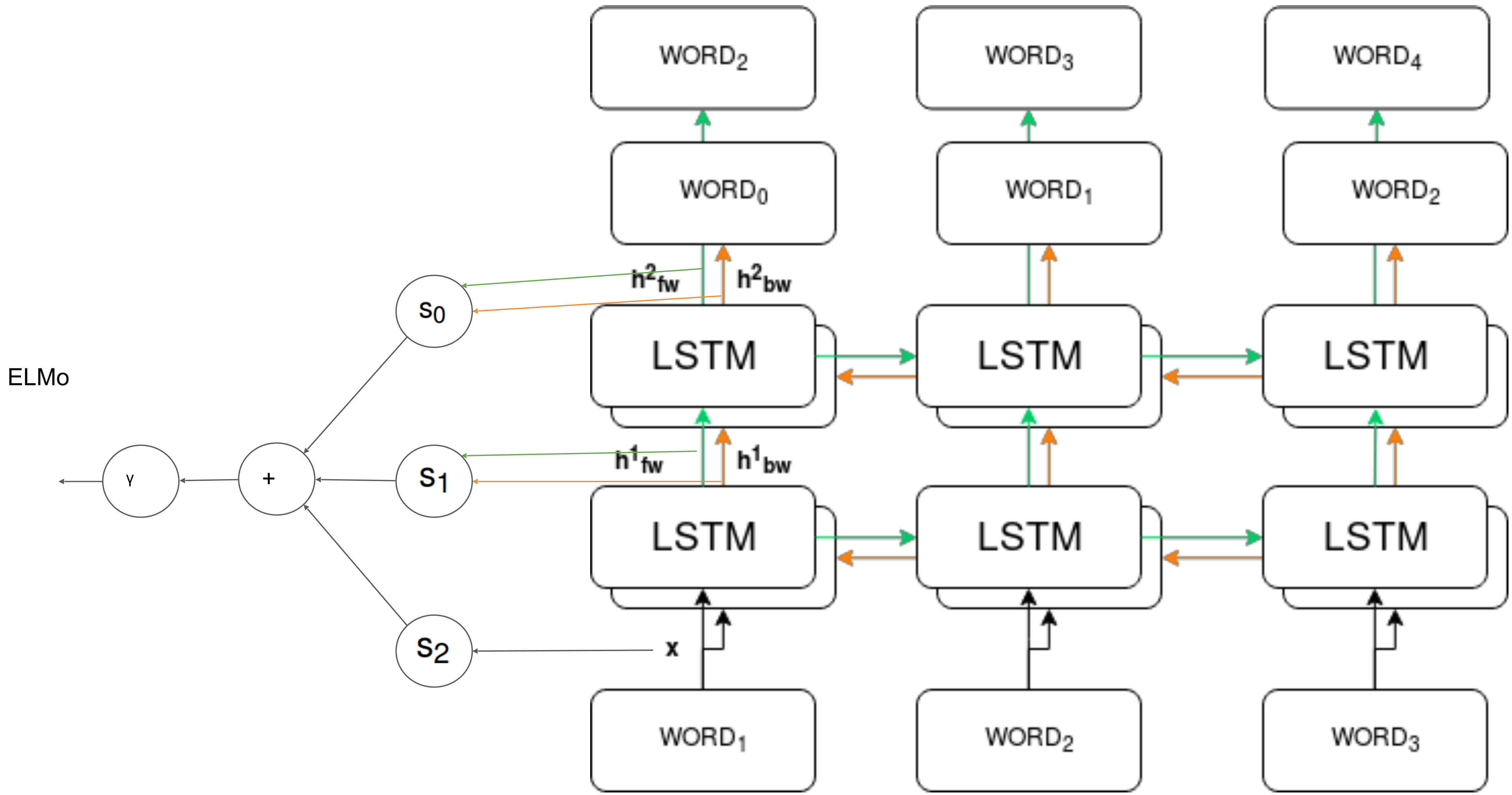
$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$

# ELMo



WORD2
WORD3
WORD4
WORD0
WORD1
WORD2
h2fw
h2bw
s0
LSTM
LSTM
LSTM
ELMo
γ
+
s1
h1fw
h1bw
LSTM
LSTM
LSTM
s2
x
WORD1
WORD2
WORD3

# ELMo




$$R_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} = \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},$$

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

# ELMo

## Results

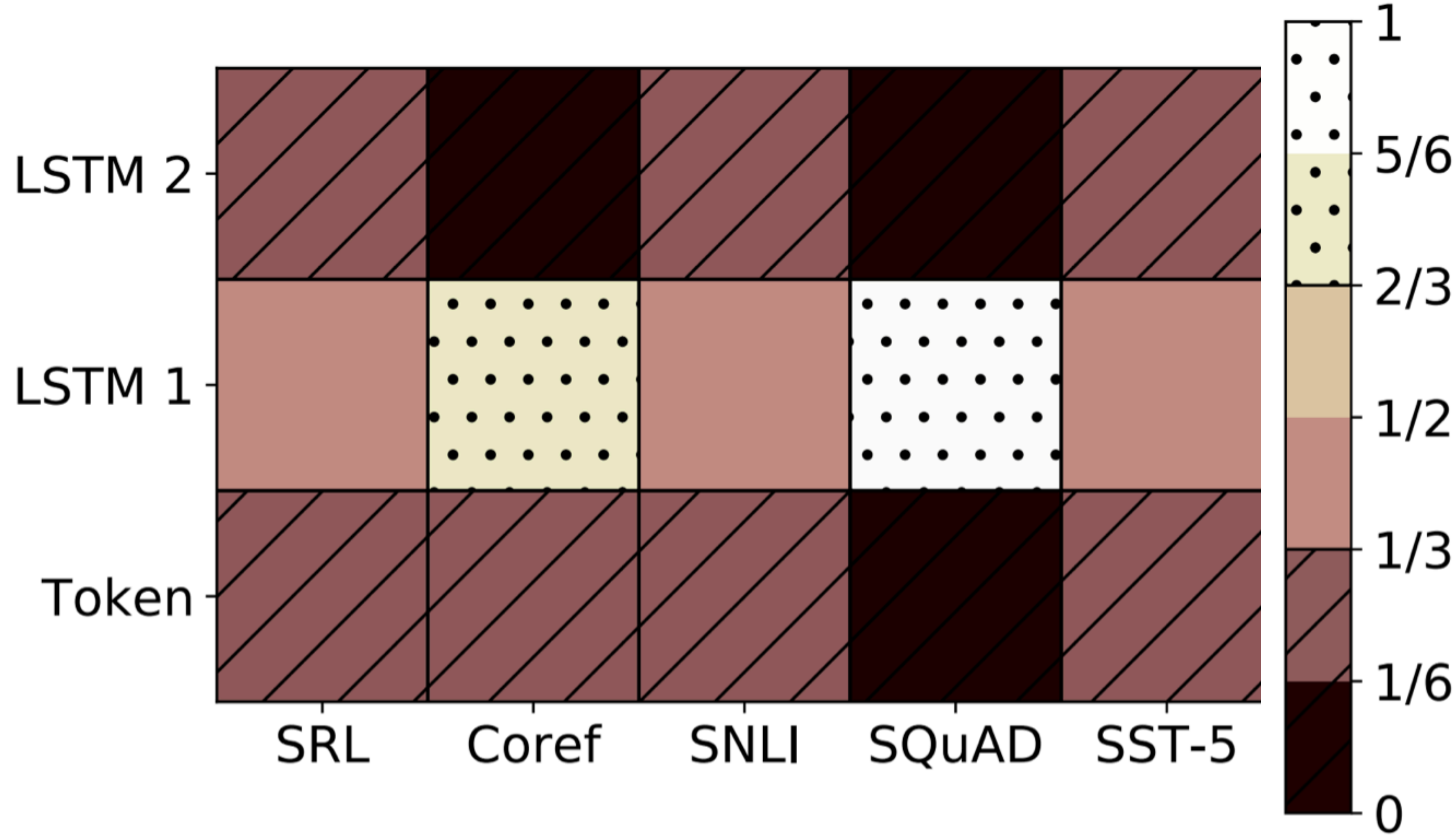
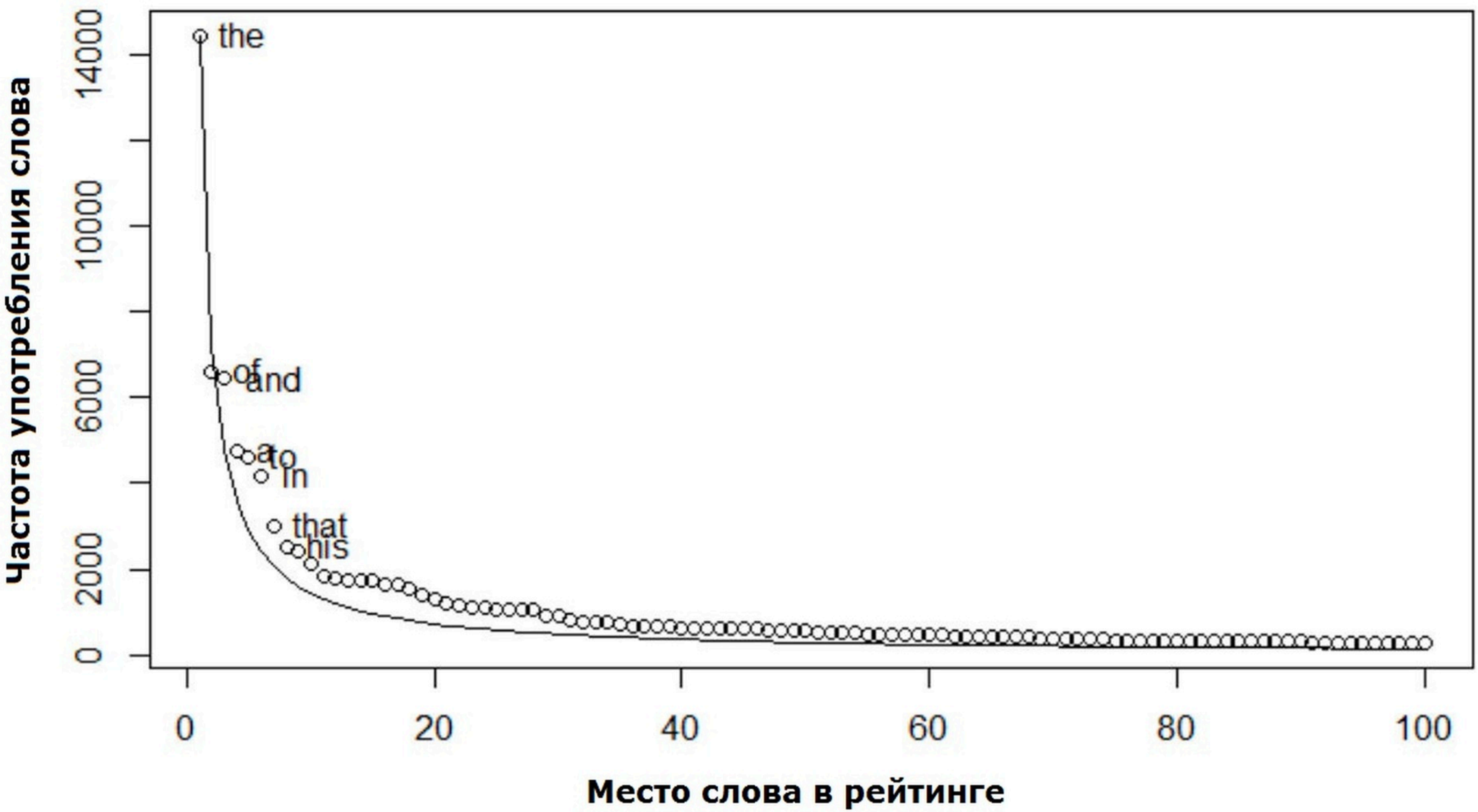



Figure 2: Visualization of softmax normalized biLM layer weights across tasks and ELMo locations. Normalized weights less then $1/3$ are hatched with horizontal lines and those greater then $2/3$ are speckled.

# Tokenization

**Закон Ципфа**



# Tokenization

**Word level**

+ Small text length
- Big vocabulary size
- OOV

**Character level**

- Long text
+ Small vocabulary size
+ Almost no OOV

# Tokenization

**Word level**

+ Small text length
- Big vocabulary size
- OOV

**Character level**

- Long text
+ Small vocabulary size
+ Almost no OOV

1. Word level

i'm a second year student in an ivy league school ->

["i'm", 'a', 'second', 'year', 'student', 'in', 'an', 'ivy', 'league', ‘school’]

2. Character level

['i', "'", 'm', ' ', 'a', ' ', 's', 'e', 'c', 'o', 'n', 'd', ' ', 'y', 'e', 'a', 'r', ' ', 's', 't', 'u', 'd', 'e', 'n', 't', ' ', 'i', 'n', ' ', 'a', 'n', ' ', 'i', 'v', 'y', ' ', 'l', 'e', 'a', 'g', 'u', 'e', ' ', 's', 'c', 'h', 'o', 'o', 'l']

# Tokenization

**Word level**

+ Small text length
- Big vocabulary size
- OOV

**Character level**

- Long text
+ Small vocabulary size
+ Almost no OOV

# BPE

I saw a girl with a telescope. ->

['▁I', '▁saw', '▁a', '▁girl', '▁with', '▁a', '▁', 'te', 'le', 's', 'c', 'o', 'pe', ‘.’]

опубликовано видео убитого саудовского журналиста джамаля хашкуджи ->

['▁опубликовано', '▁видео', '▁убитого', '▁саудов', 'ского', '▁журналиста', '▁джама', 'ля', '▁ха', 'шку', 'джи']

# BPE

**Algorithm 1** Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

| | | |
|---|---|---|
| r · | → | r· |
| l o | → | lo |
| lo w | → | low |
| e r· | → | er· |

Figure 1: BPE merge operations learned from dictionary {‘low’, ‘lowest’, ‘newer’, ‘wider’}.

- learning
  - word:freq : {low:5, lowest:2, newer:6, wider:3}
  - marge & count
    1. ‘r’ ‘</w>’ : 9 → marge’r</w>’
    2. ‘e’ ‘r</w>’ : 9 →marge’er</w>’
    3. ‘l’ ‘o’ : 7 →marge’lo’
    4. ‘lo’ ‘w’ : 7 →marge’low’

→ OOV : ‘lower’ segmented ‘low er</w>’

**Vocabulary sizes:**
5000, 10000, 15000, …, 50000

# BPE

## SentencePiece

build failing | build passing | coverage 98% | issues 35 open | code quality A | pypi package 0.1.83 | contributions welcome | License Apache 2.0

SentencePiece is an unsupervised text tokenizer and detokenizer mainly for Neural Network-based text generation systems where the vocabulary size is predetermined prior to the neural model training. SentencePiece implements **subword units** (e.g., **byte-pair-encoding (BPE)** [Sennrich et al.]) and **unigram language model** [Kudo.]) with the extension of direct training from raw sentences. SentencePiece allows us to make a purely end-to-end system that does not depend on language-specific pre/postprocessing.

**This is not an official Google product.**

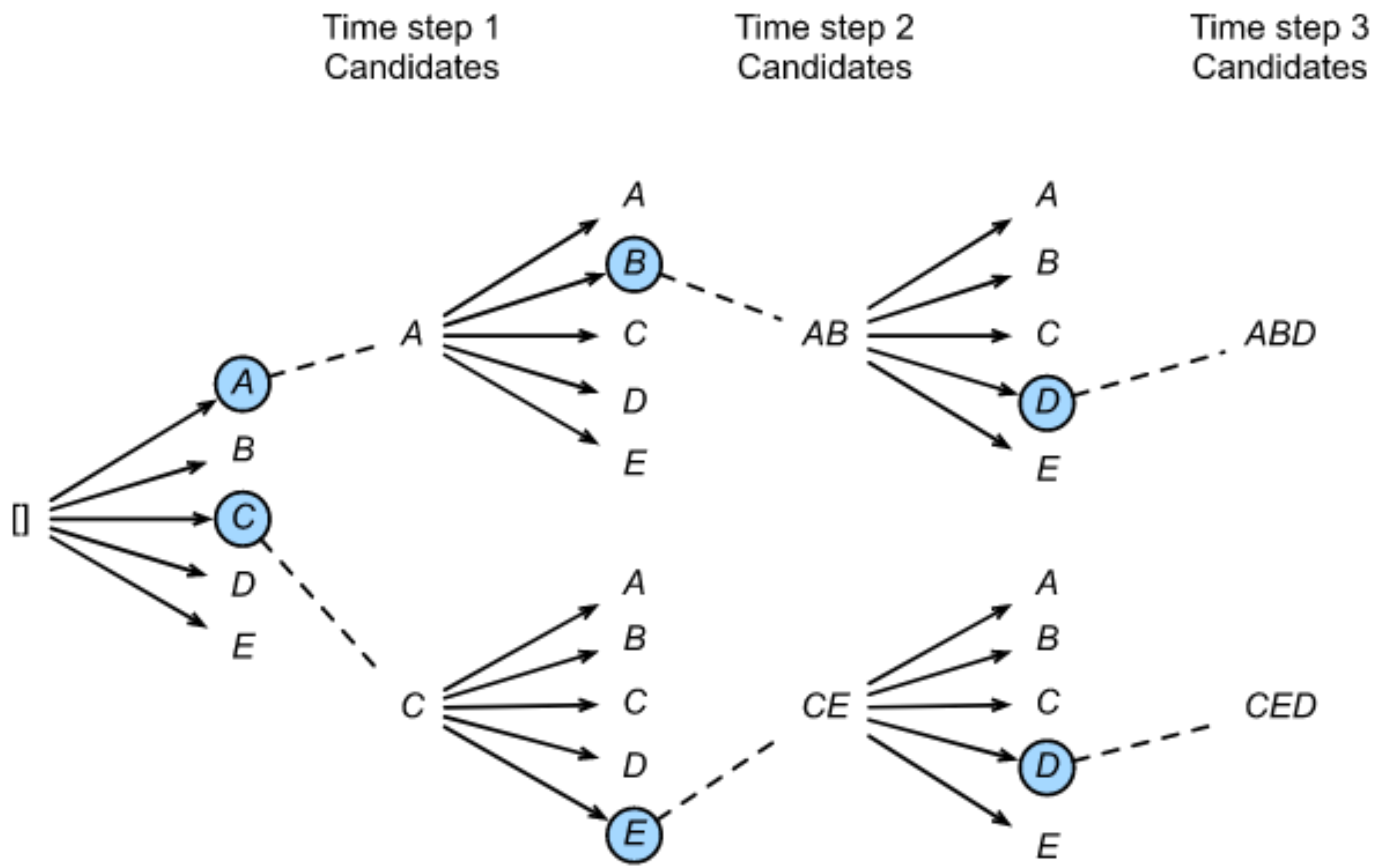
## YouTokenToMe

YouTokenToMe is an unsupervised text tokenizer focused on computational efficiency. It currently implements fast Byte Pair Encoding (BPE) [Sennrich et al.]. Our implementation is much faster in training and tokenization than both fastBPE and SentencePiece. In some test cases, it is 90 times faster. Check out our benchmark results.

Key advantages:

- Multithreading for training and tokenization
- The algorithm has `O(N)` complexity, where `N` is the length of training data
- Highly efficient implementation in C++
- Python wrapper and command-line interface

# Beam Search



Time step 1
Candidates
Time step 2
Candidates
Time step 3
Candidates
[]
A
B
C
D
E
A
C
A
B
C
D
E
A
B
C
D
E
AB
CE
A
B
C
D
E
A
B
C
D
E
ABD
CED

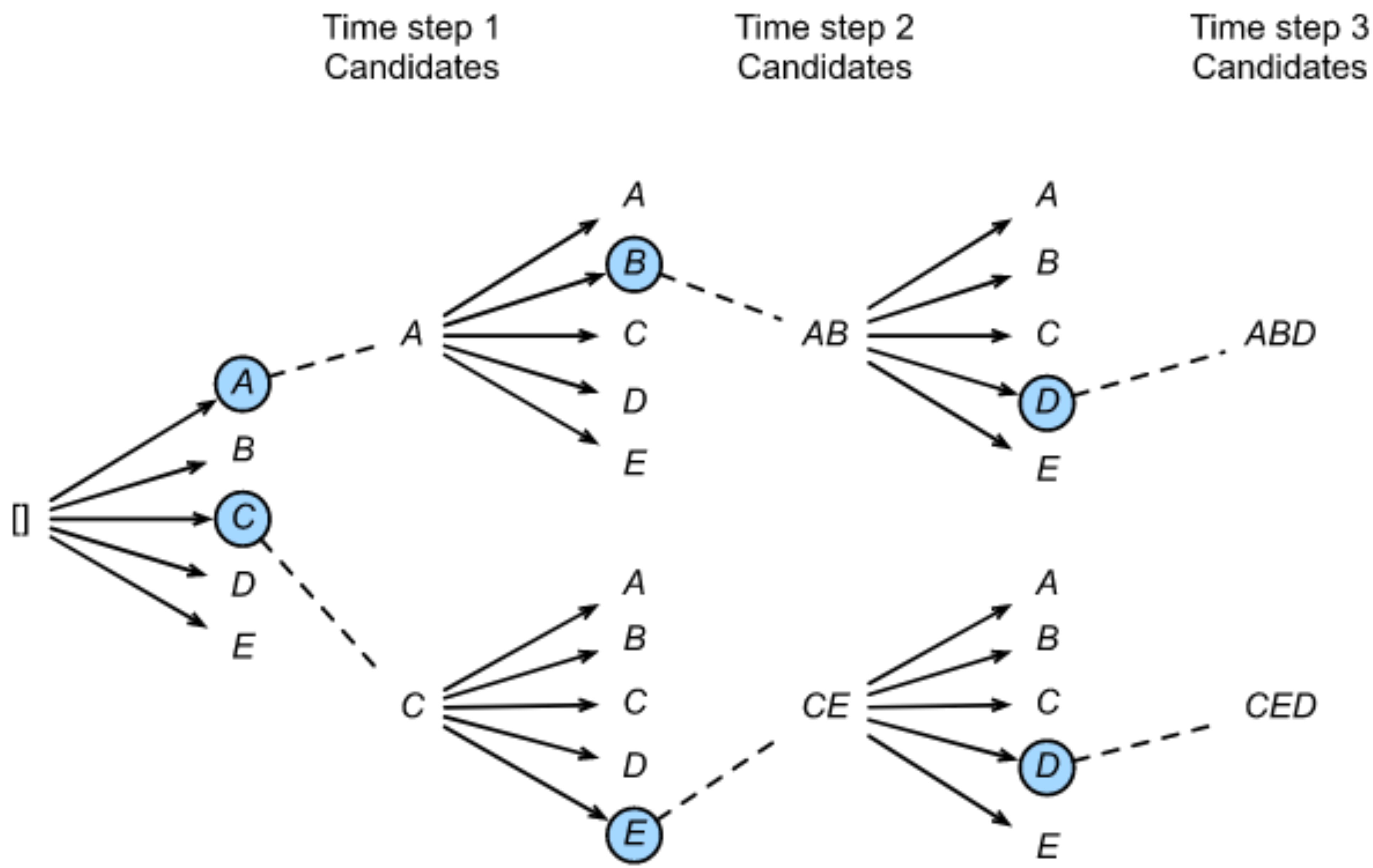
# Beam Search

$$p(\mathbf{x}) = \prod_i p(x|x_{<i}) = p(x_0)p(x_1|x_0)p(x_2|x_0,x_1)\ldots$$

$$\arg\max_y \prod_{t=1}^{T_y} P(y^{<t>} \mid x, y^{<1>}, \ldots, y^{<t-1>})$$

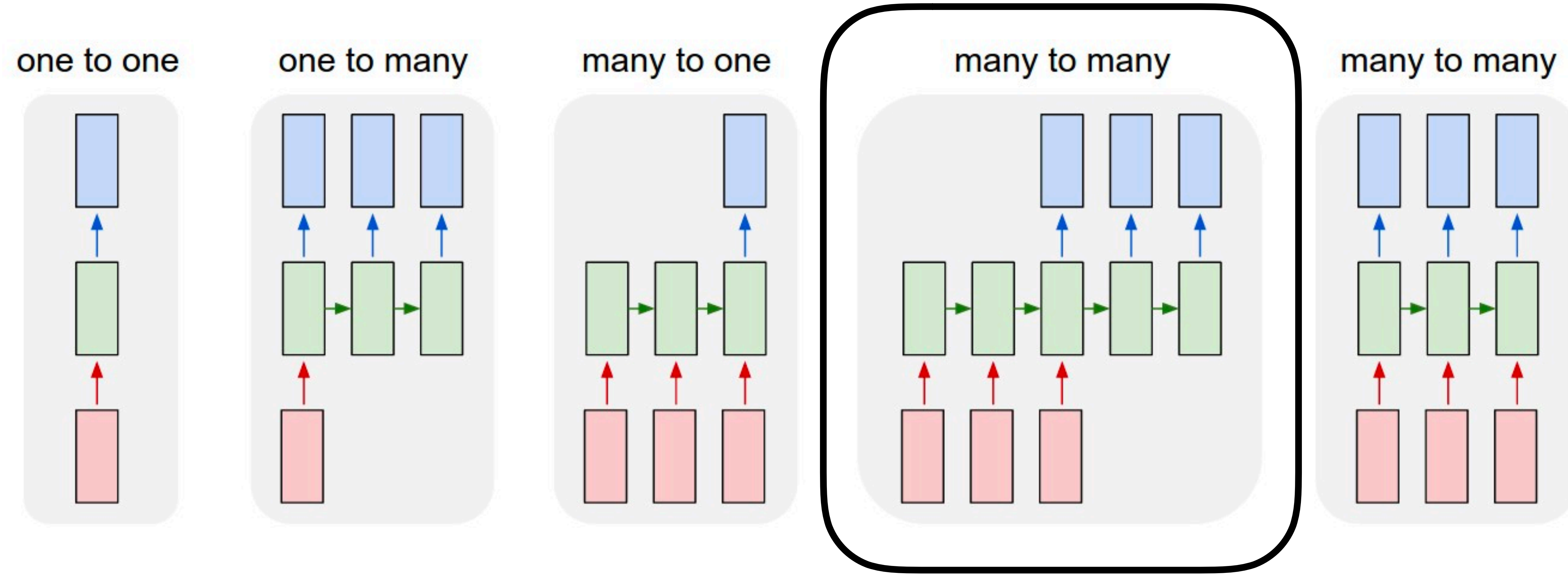
$$\arg\max_y \sum_{y=1}^{T_y} \log P(y^{<t>} \mid x, y^{<1>}, \ldots, y^{<t-1>})$$

# Beam Search



Time step 1
Candidates
Time step 2
Candidates
Time step 3
Candidates
[]
A
B
C
D
E
A
C
A
B
C
D
E
A
B
C
D
E
AB
CE
A
B
C
D
E
A
B
C
D
E
ABD
CED

# Sequence to sequence



# Sequence to sequence

## Inference