Language Models

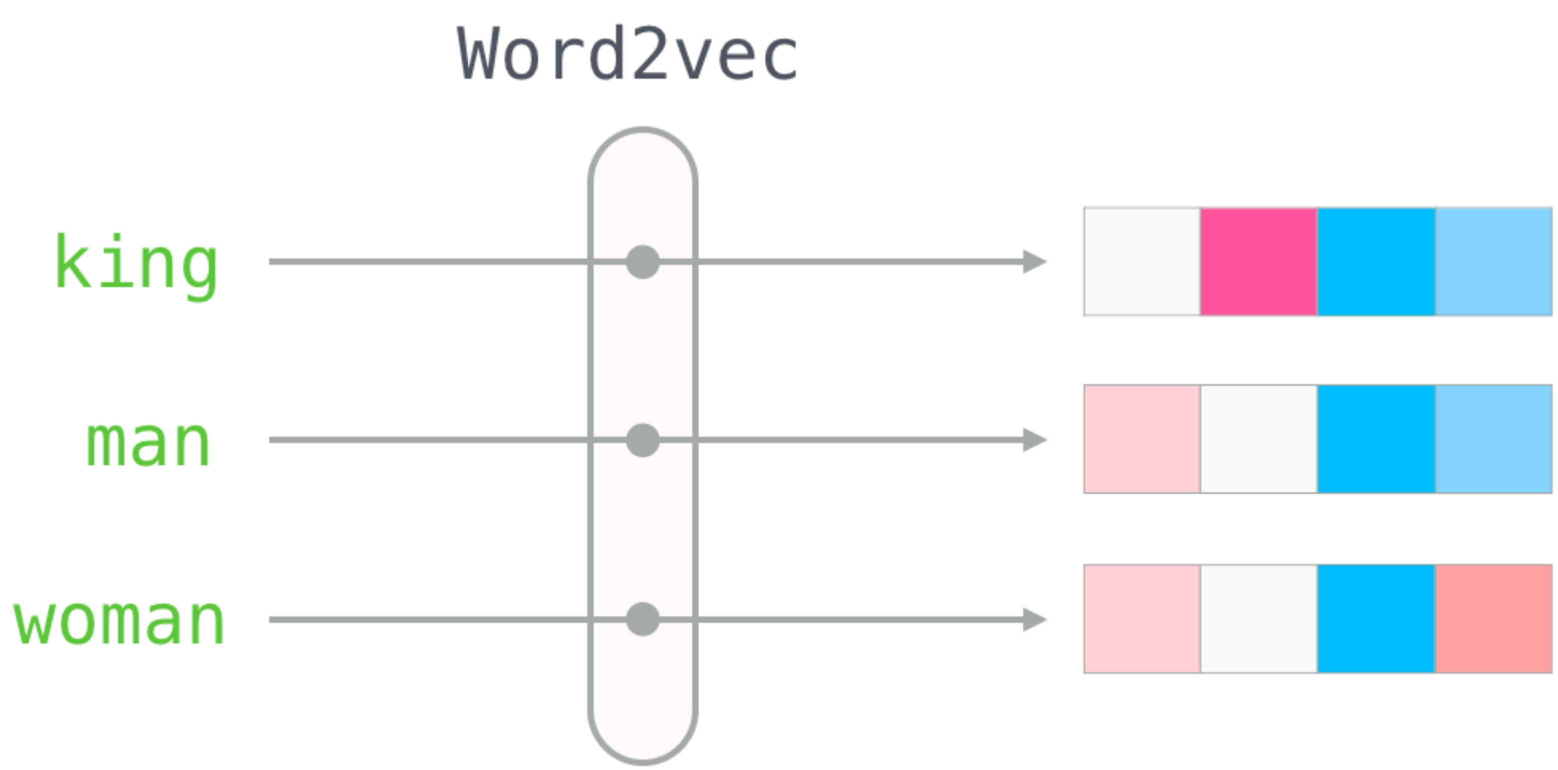
Boris Zubarev

@bobazooba

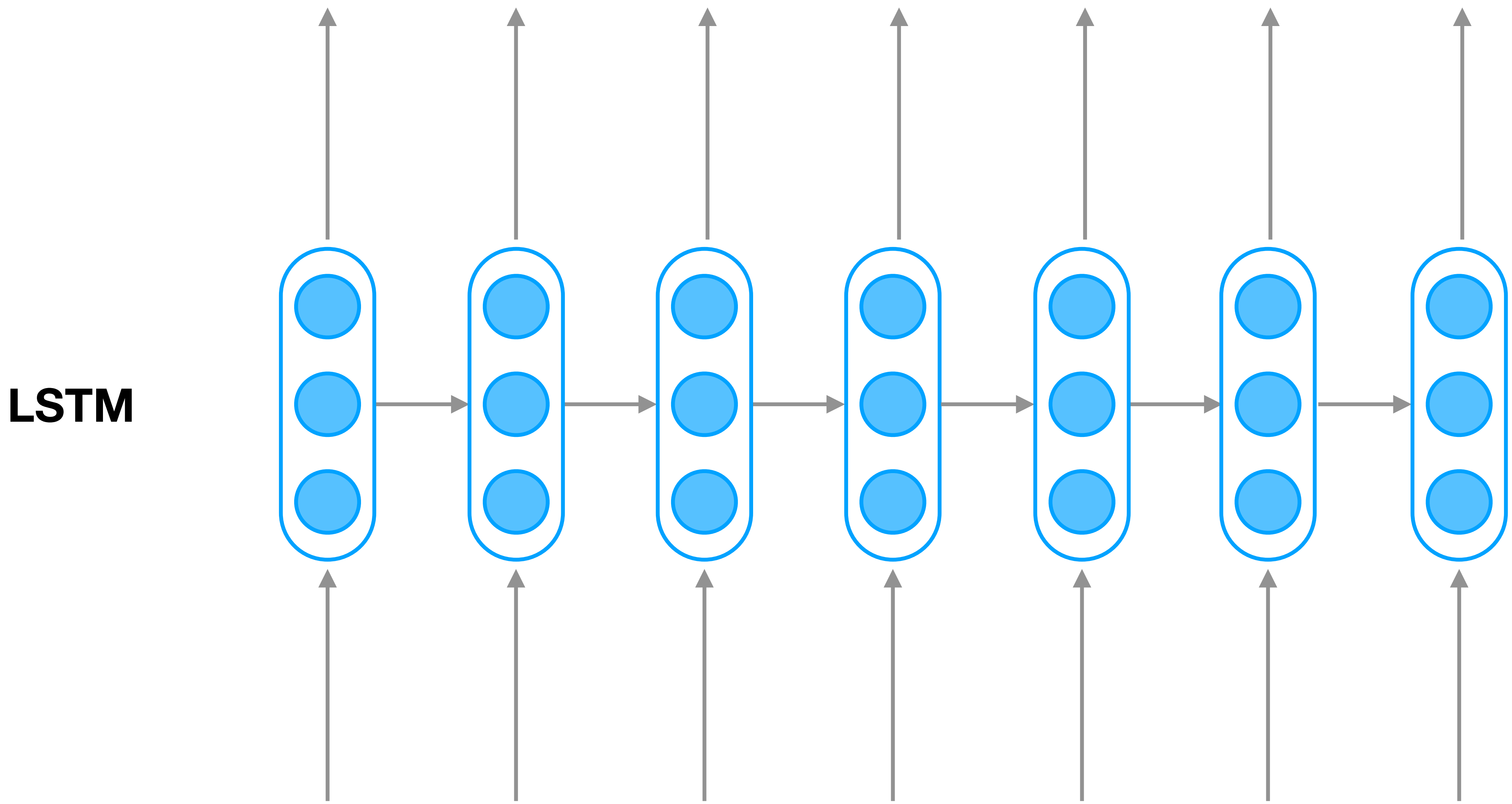
Word Embeddings

w2v, glove, etc

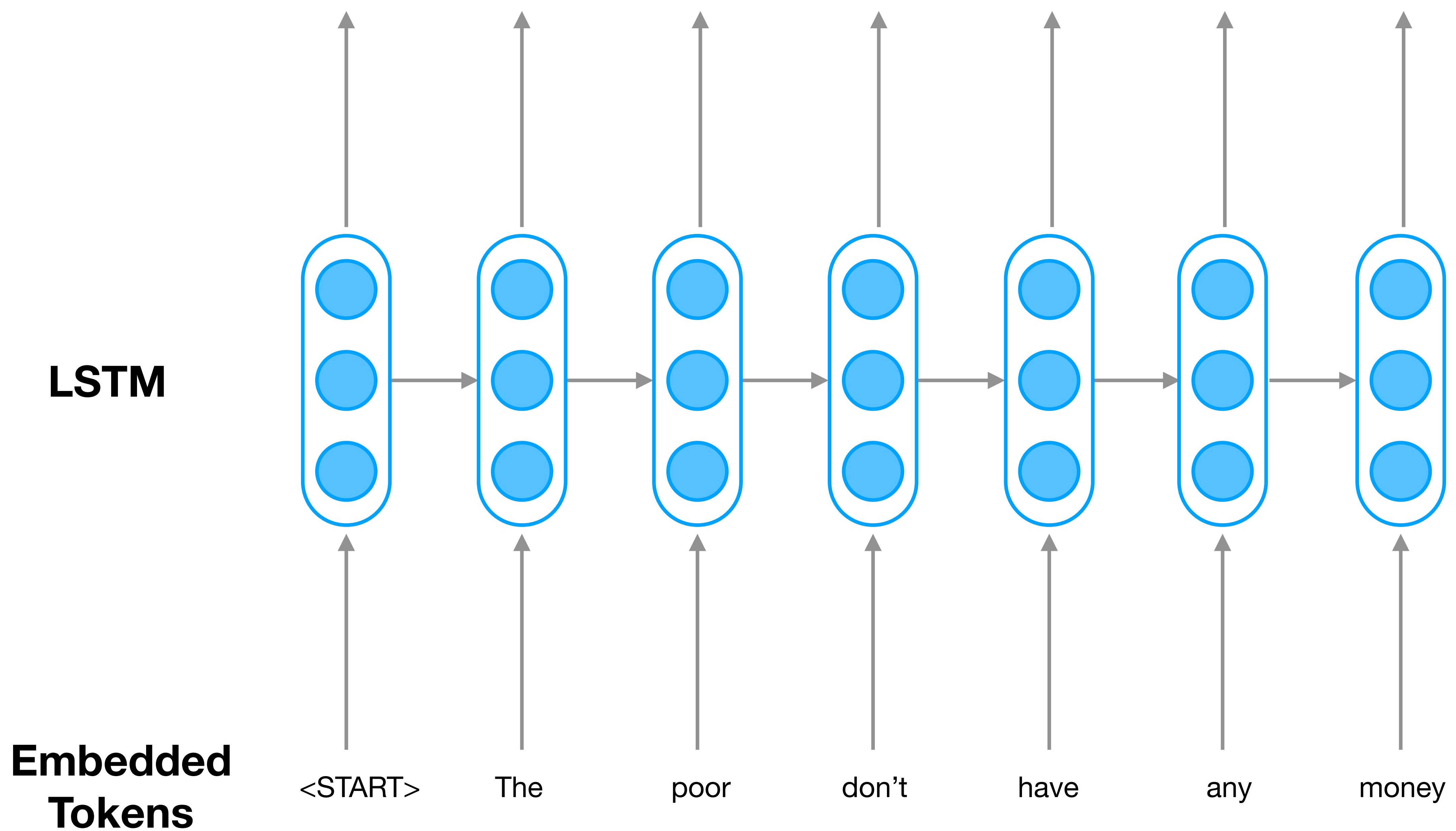
- Just key—value storage at inference
- Don’t change from relationships with other words in the current text



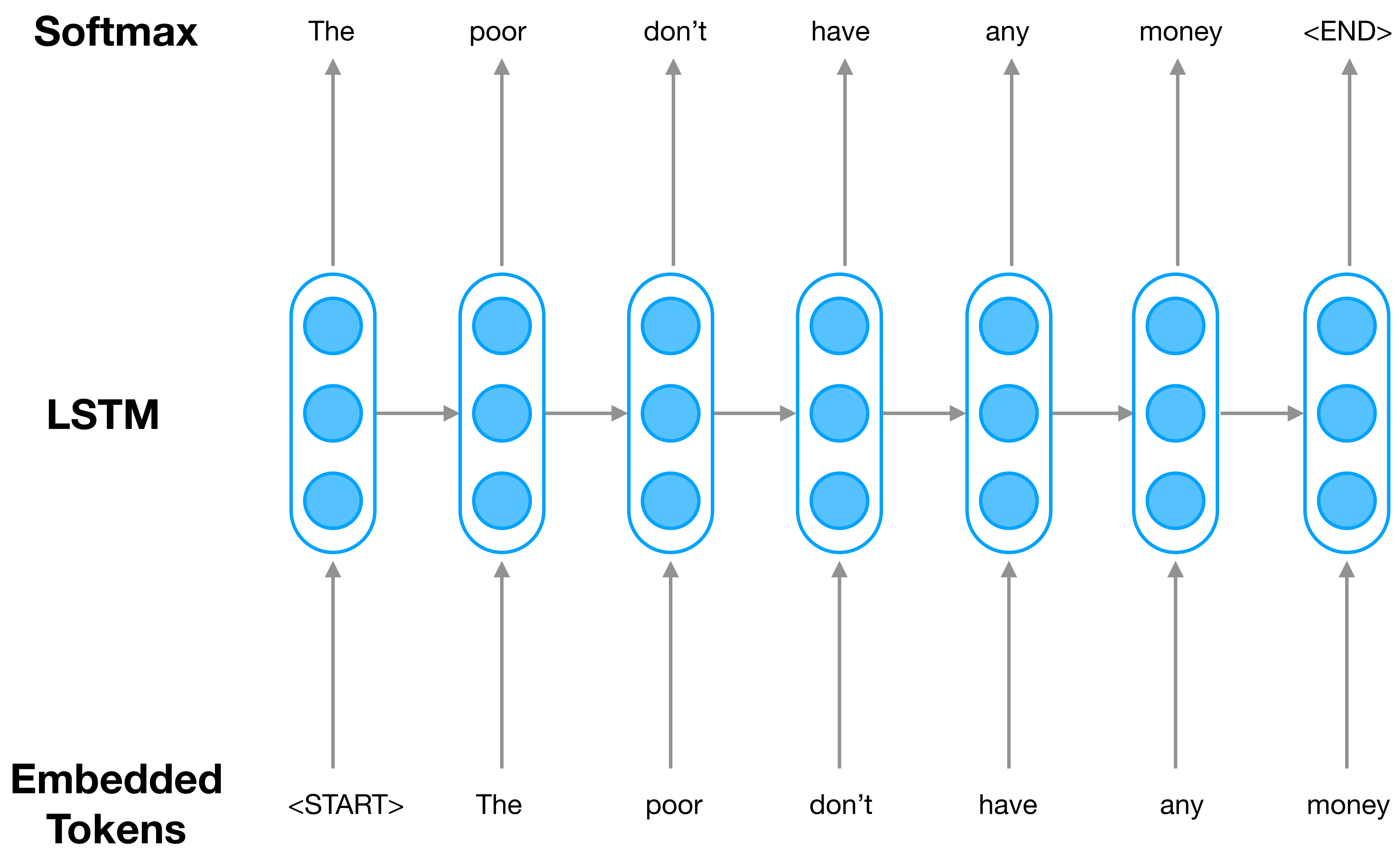
Language Model



Language Model



Language Model



Language Model

Inference

LSTM

Embedded Tokens

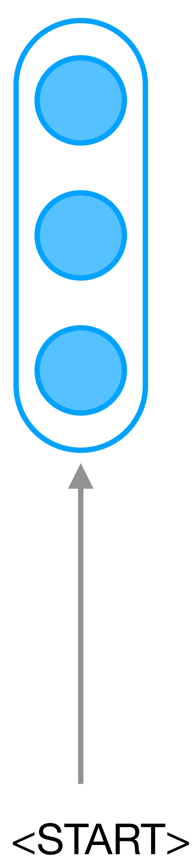
<START>

Language Model

Inference

Softmax

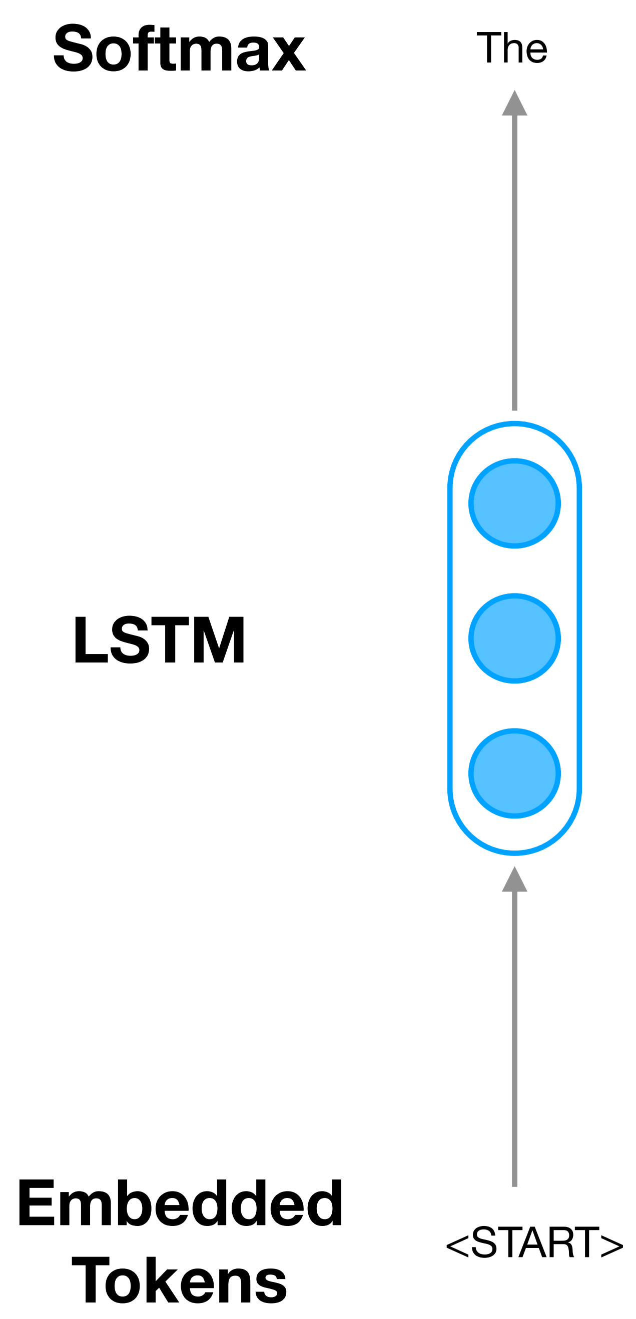
LSTM



Embedded Tokens

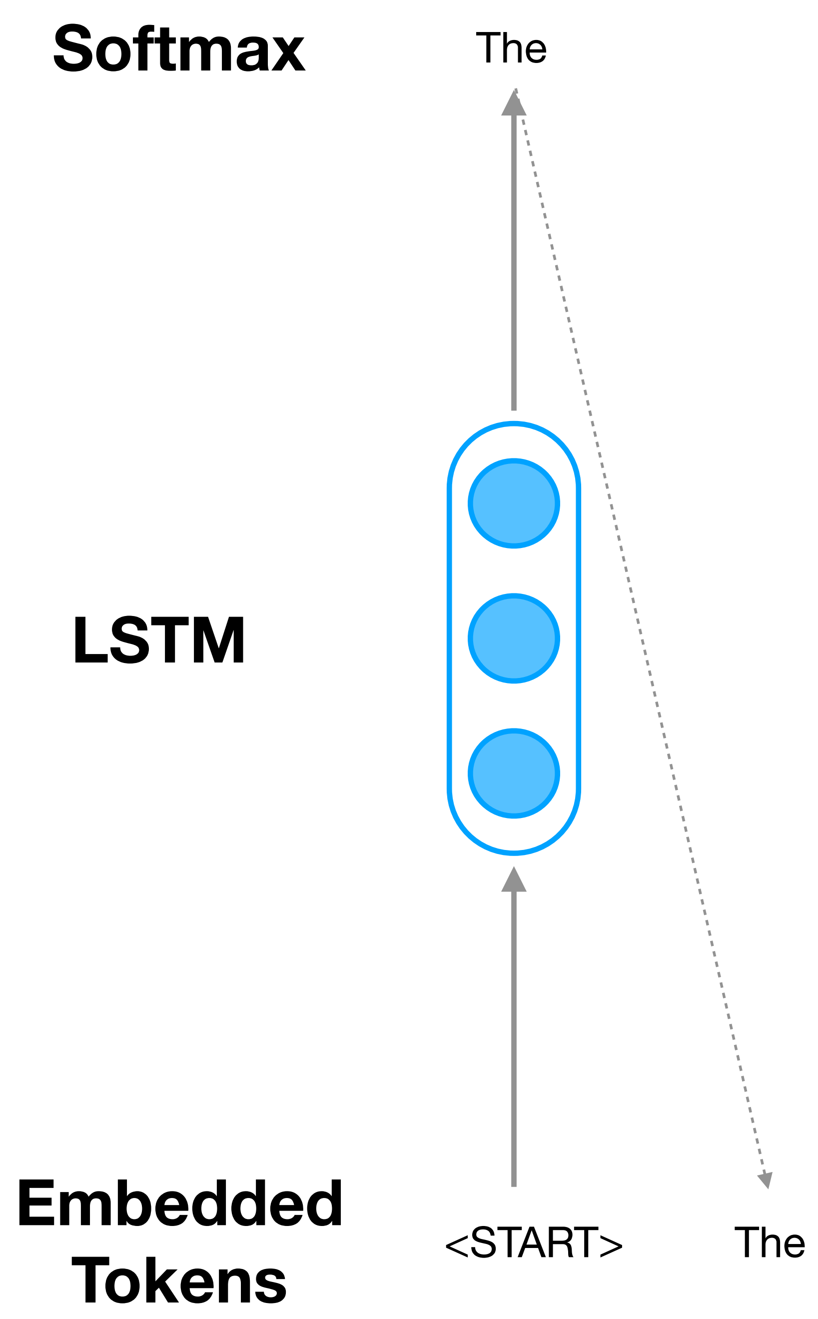
Language Model

Inference



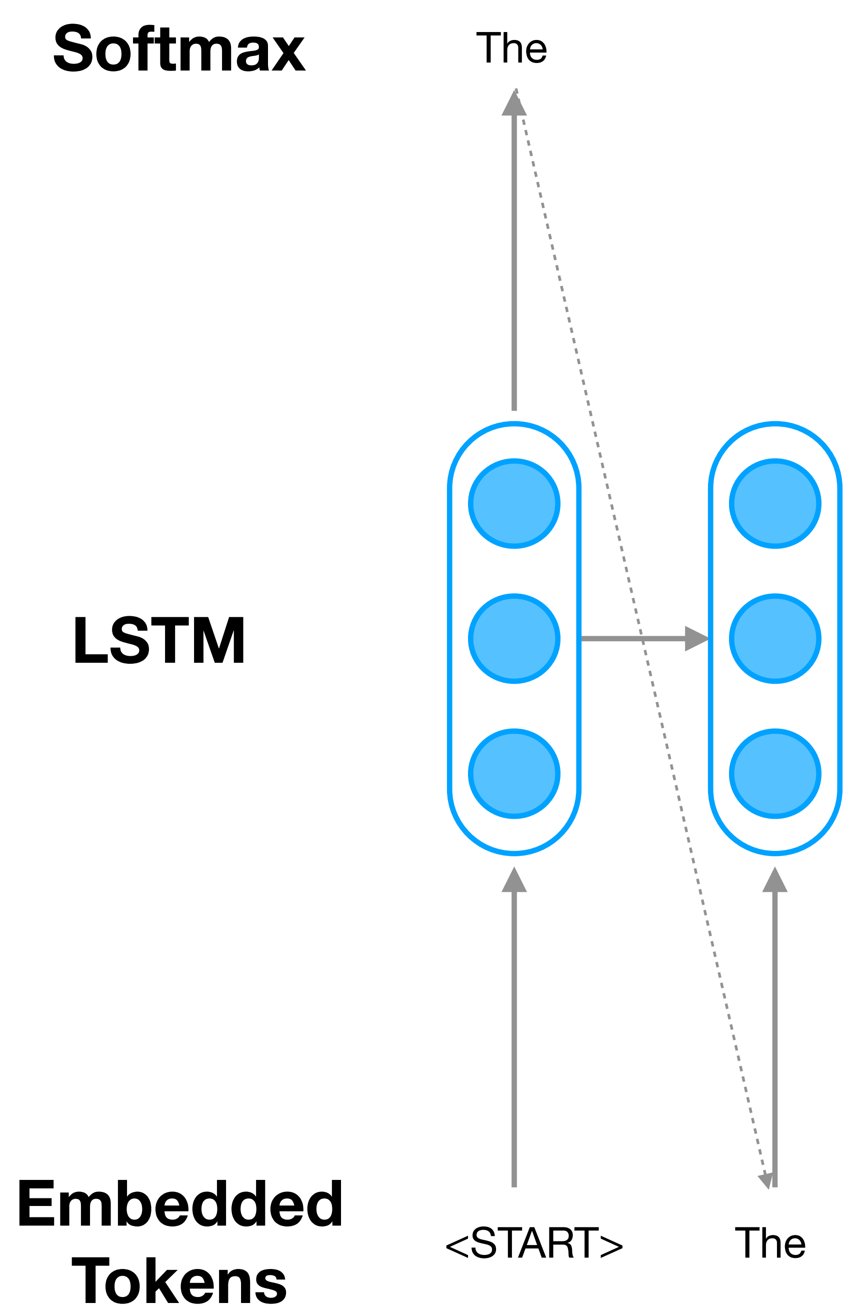
Language Model

Inference



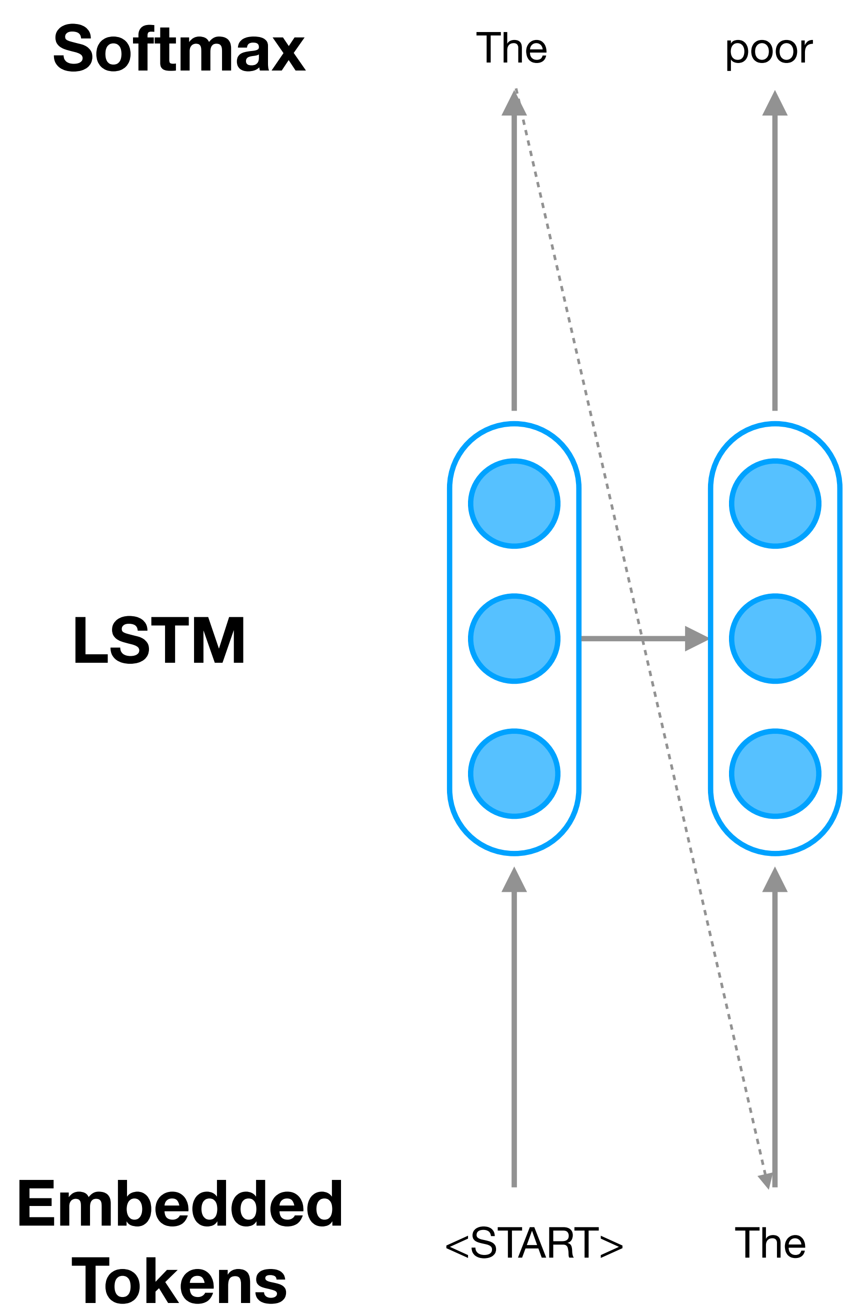
Language Model

Inference



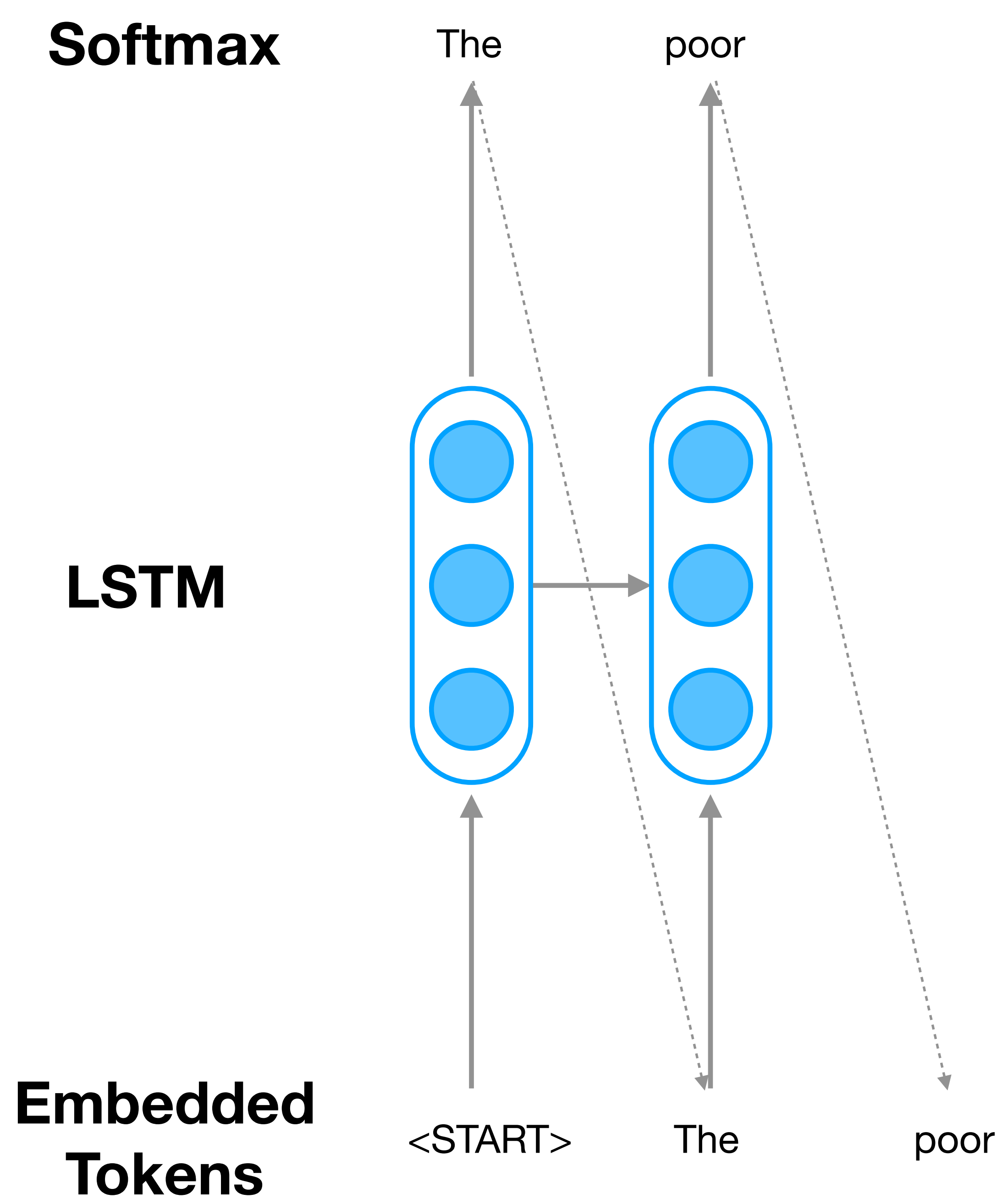
Language Model

Inference



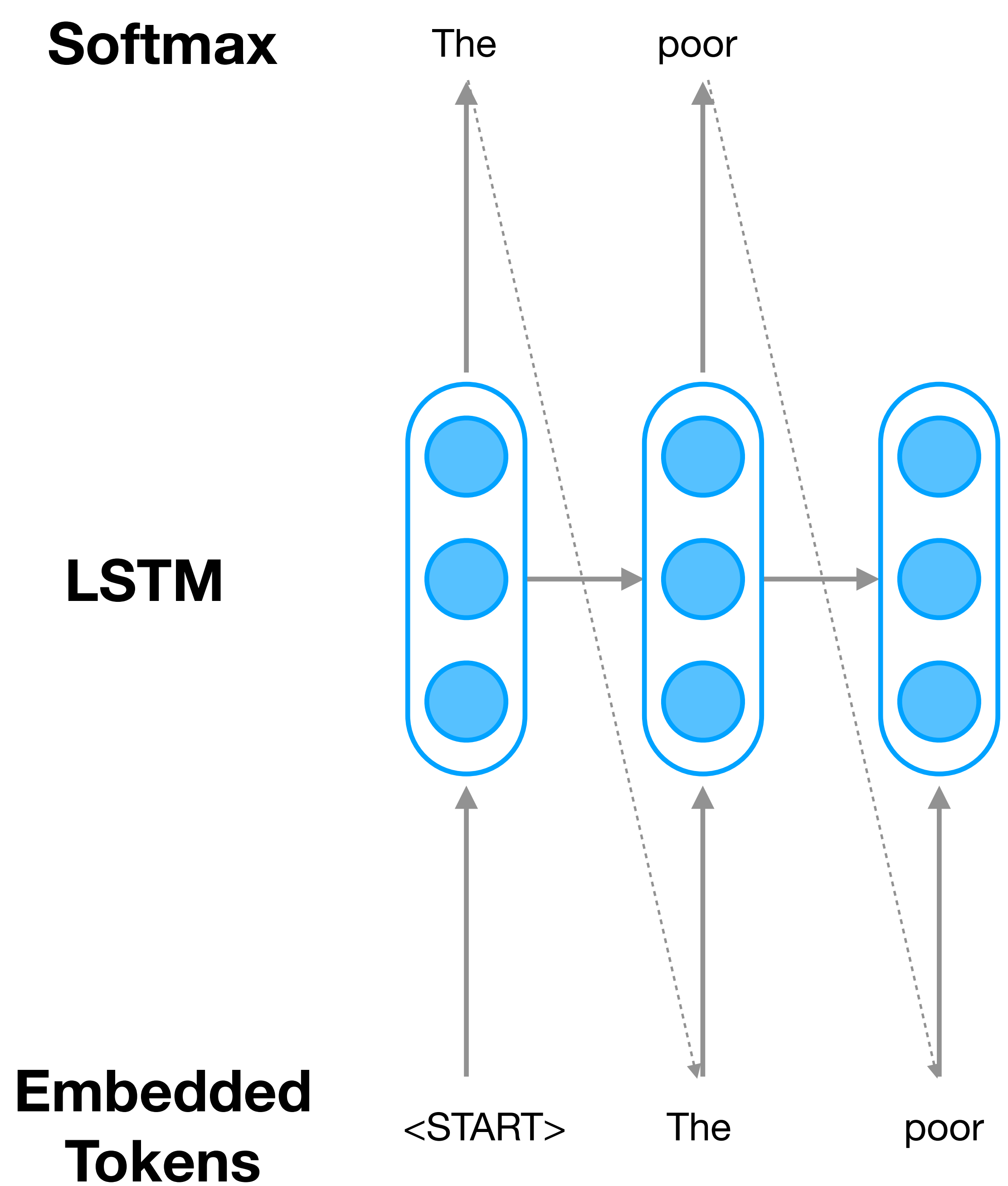
Language Model

Inference



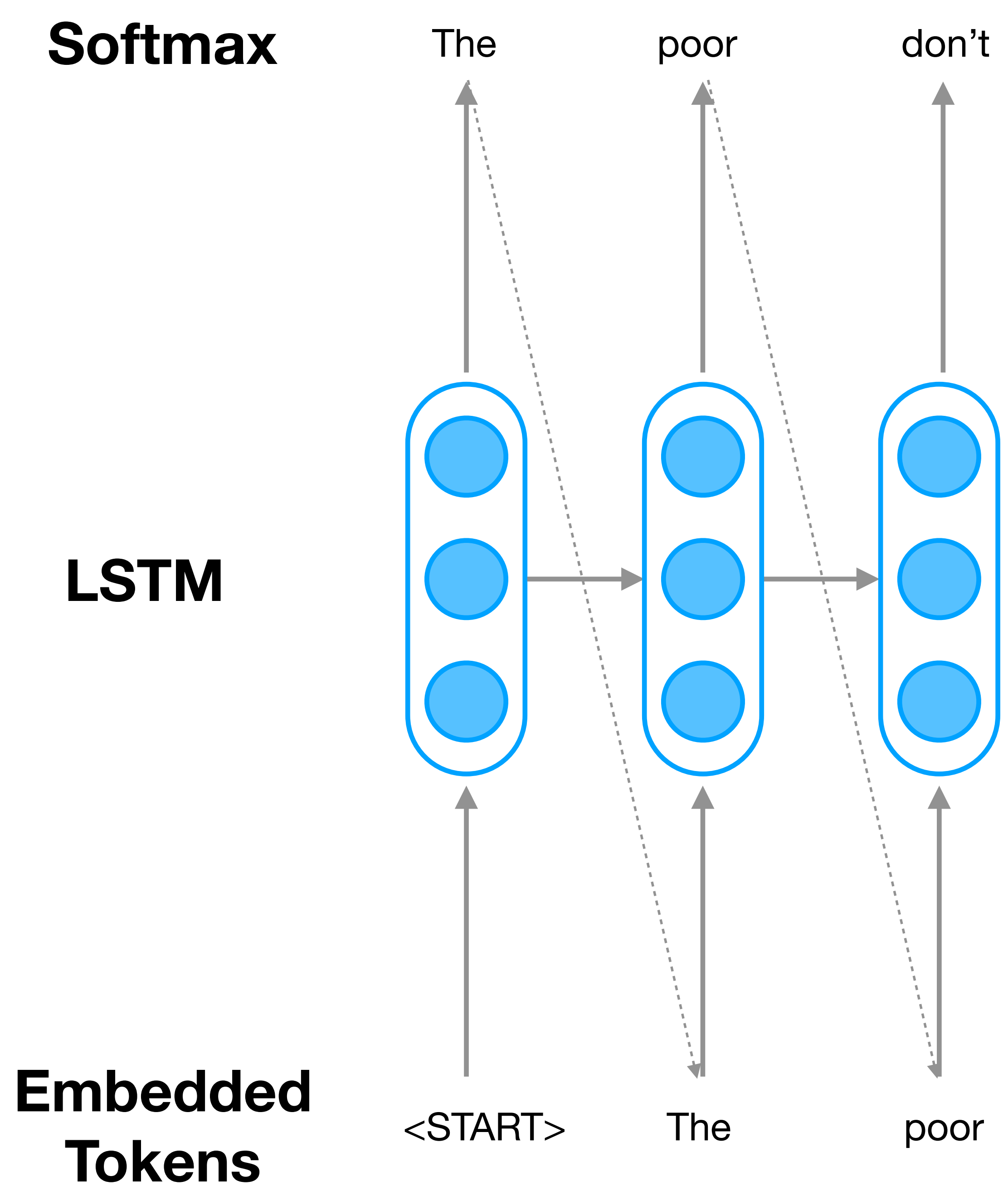
Language Model

Inference



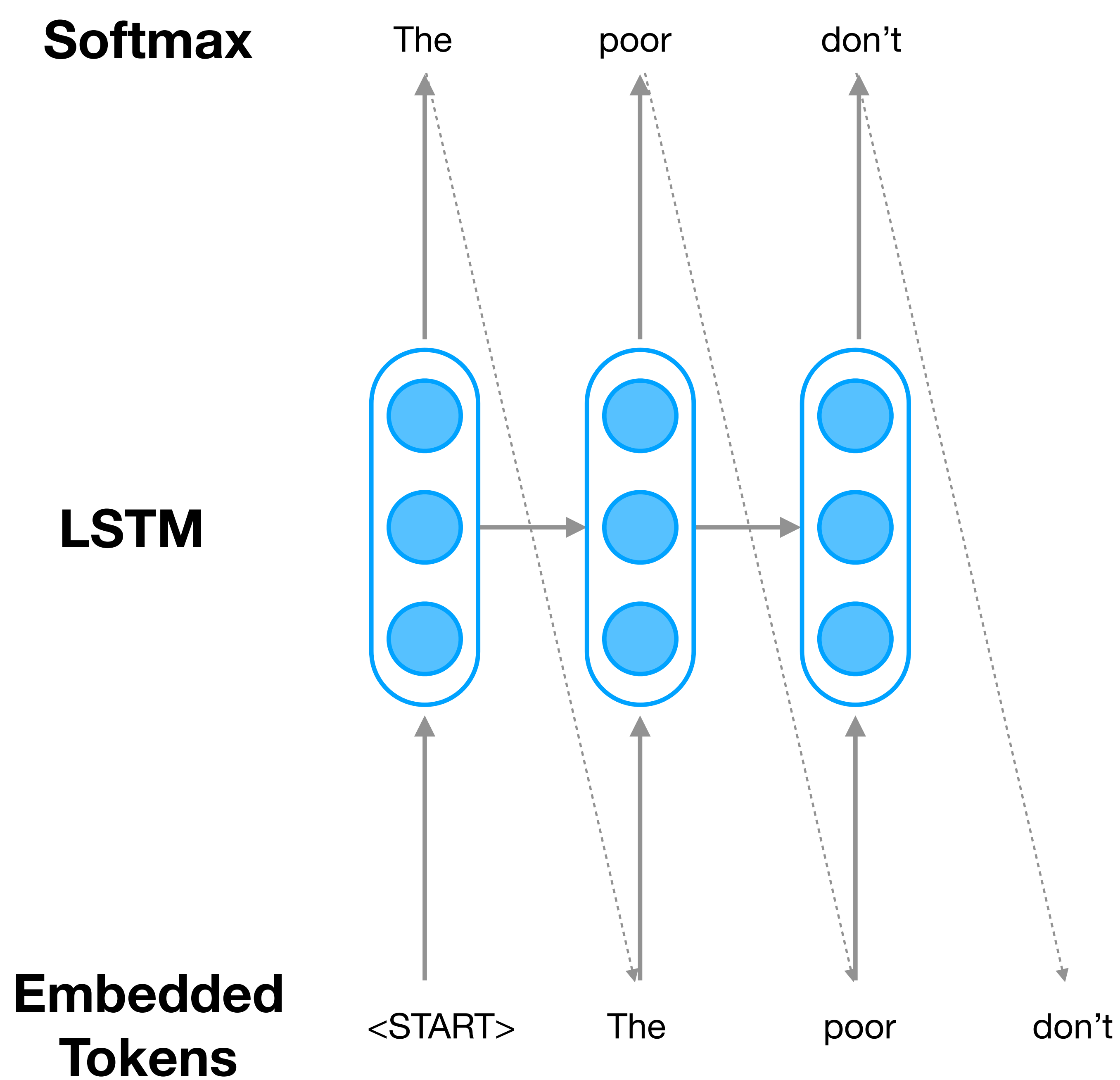
Language Model

Inference



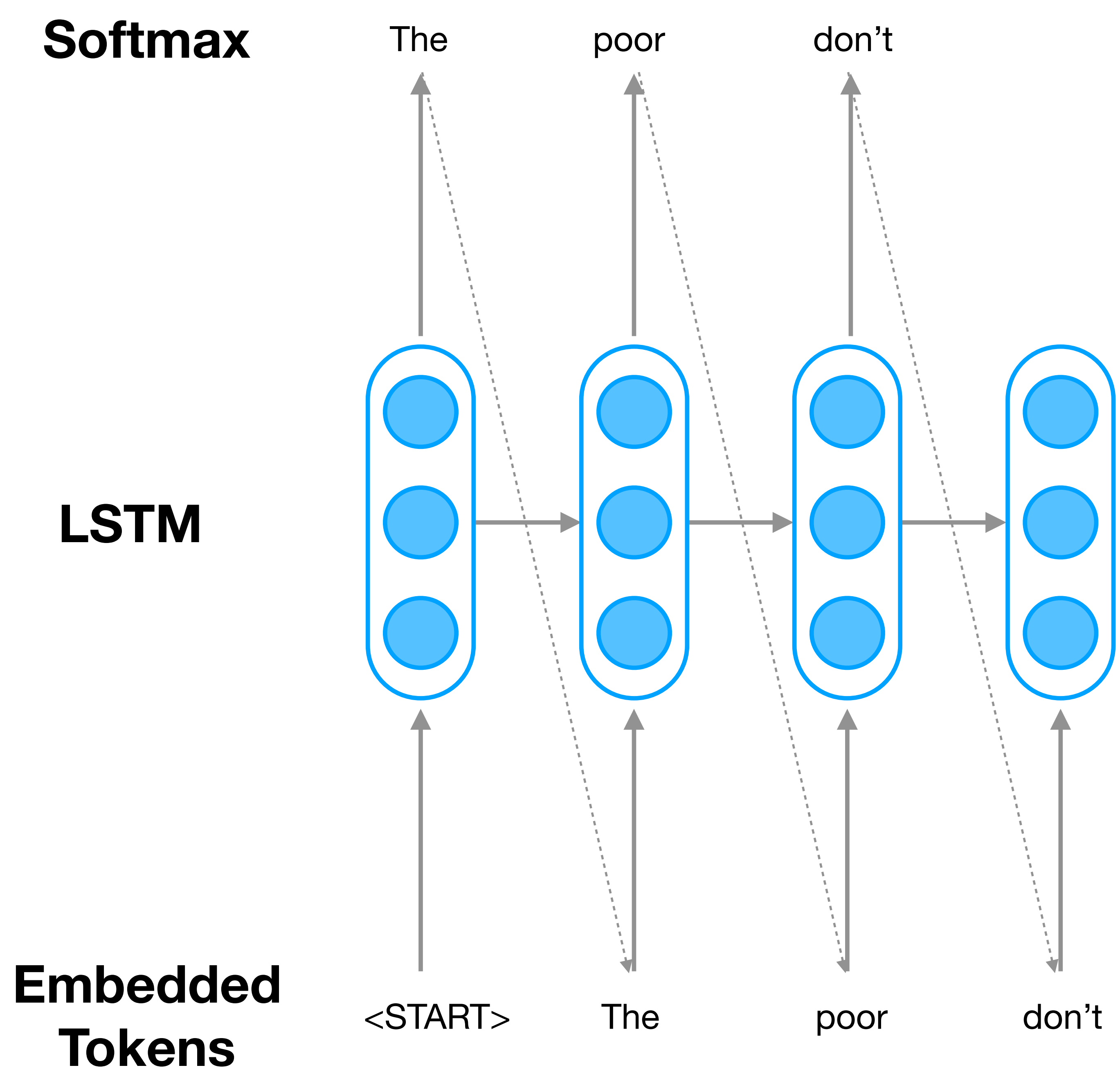
Language Model

Inference



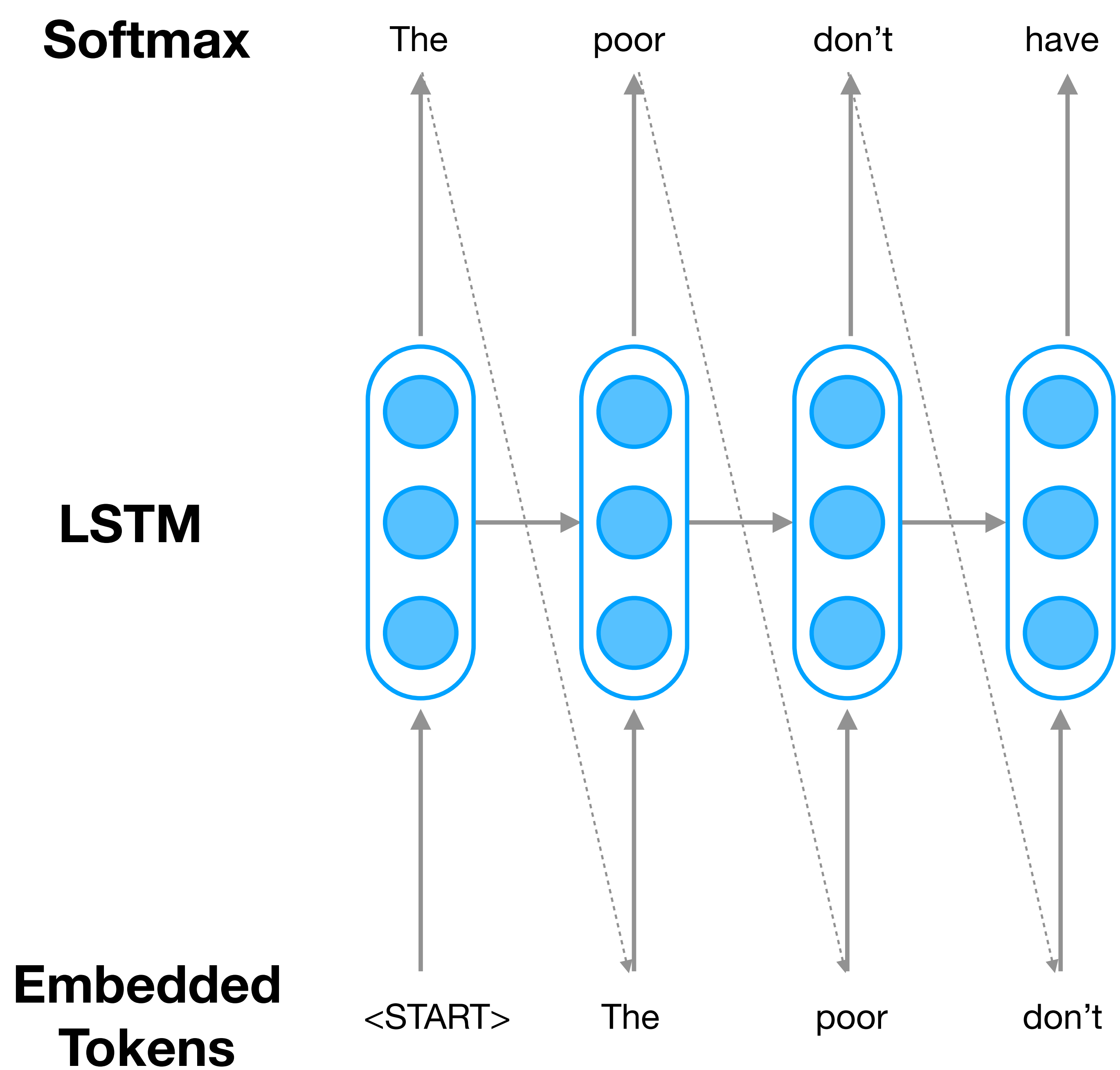
Language Model

Inference



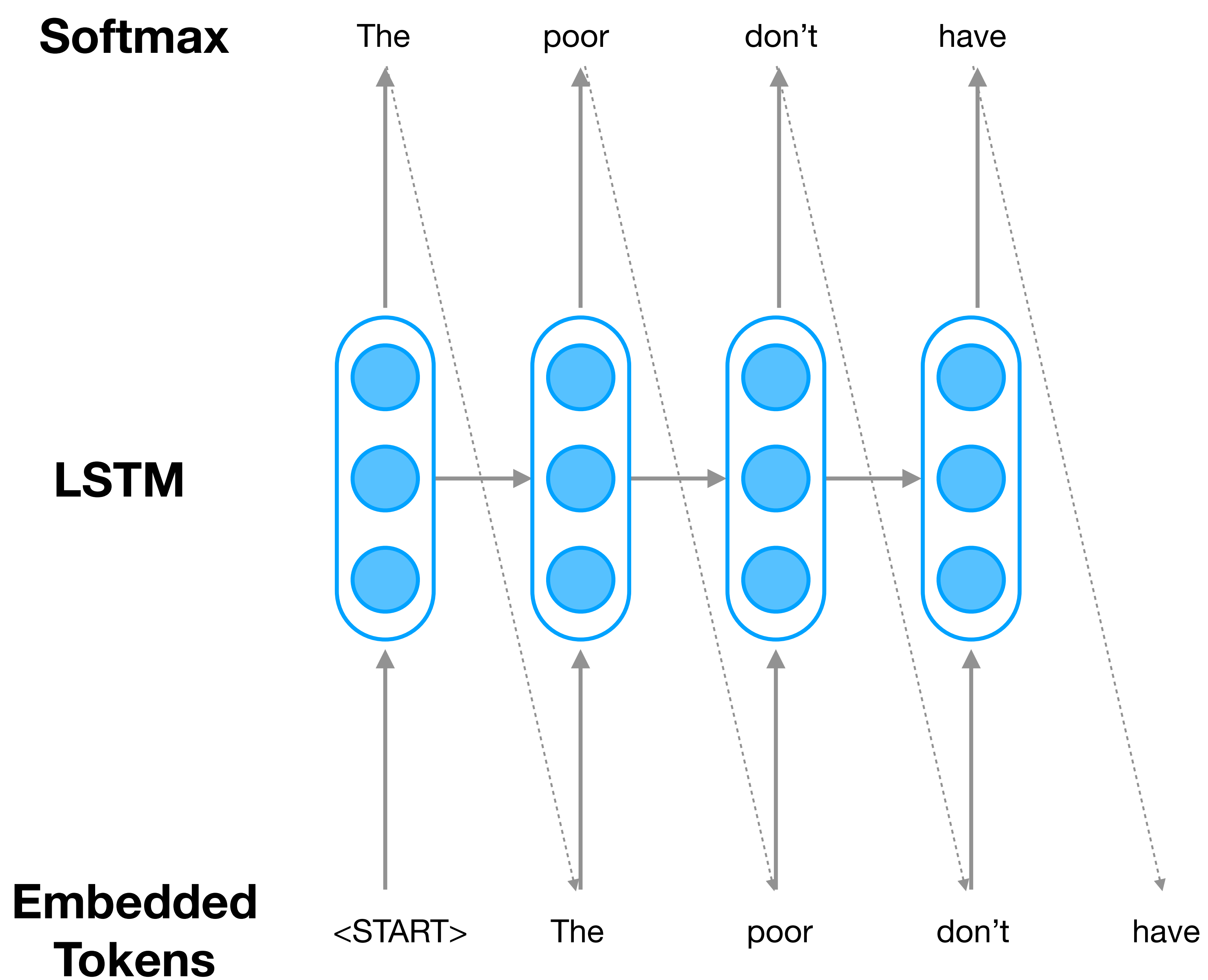
Language Model

Inference



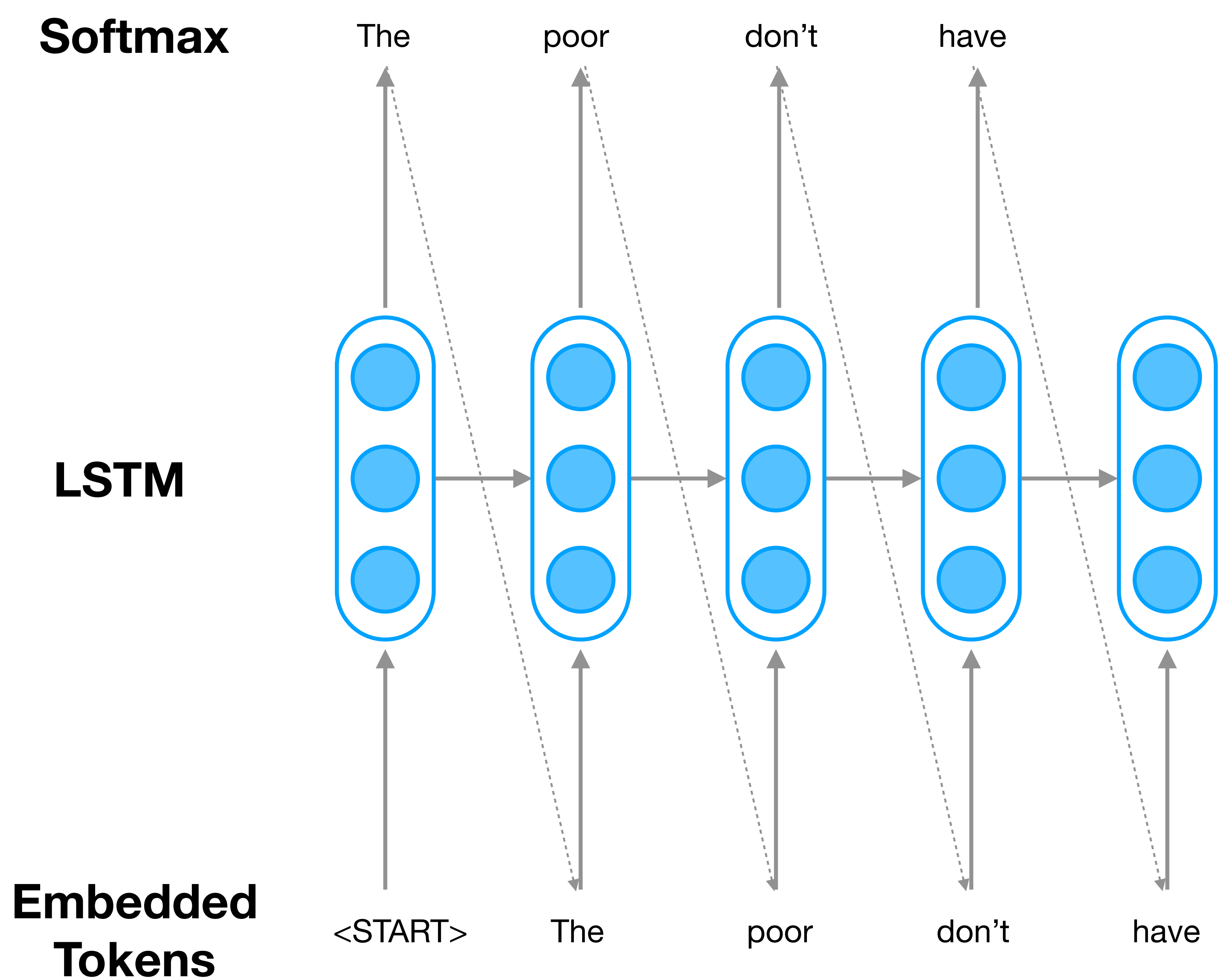
Language Model

Inference



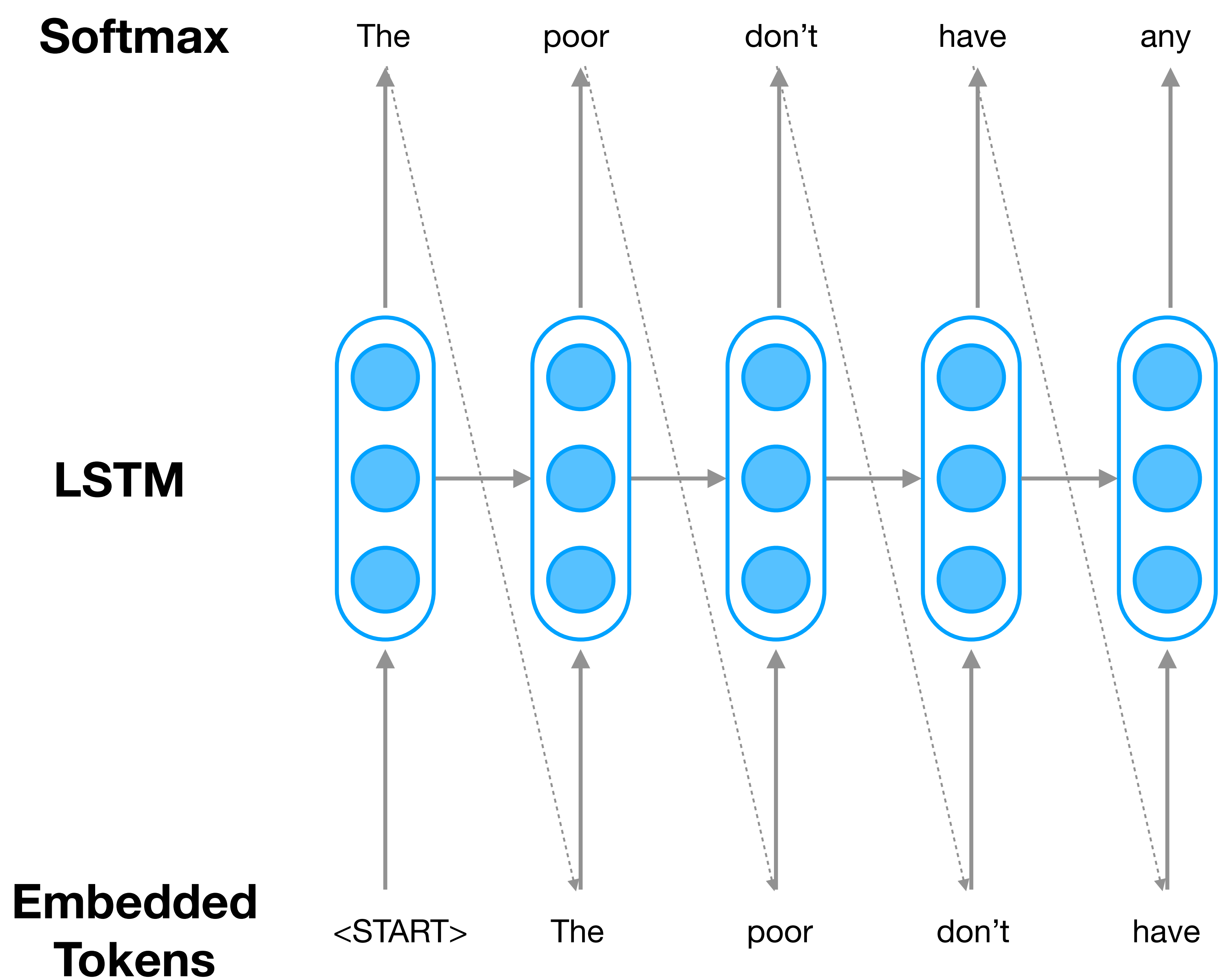
Language Model

Inference



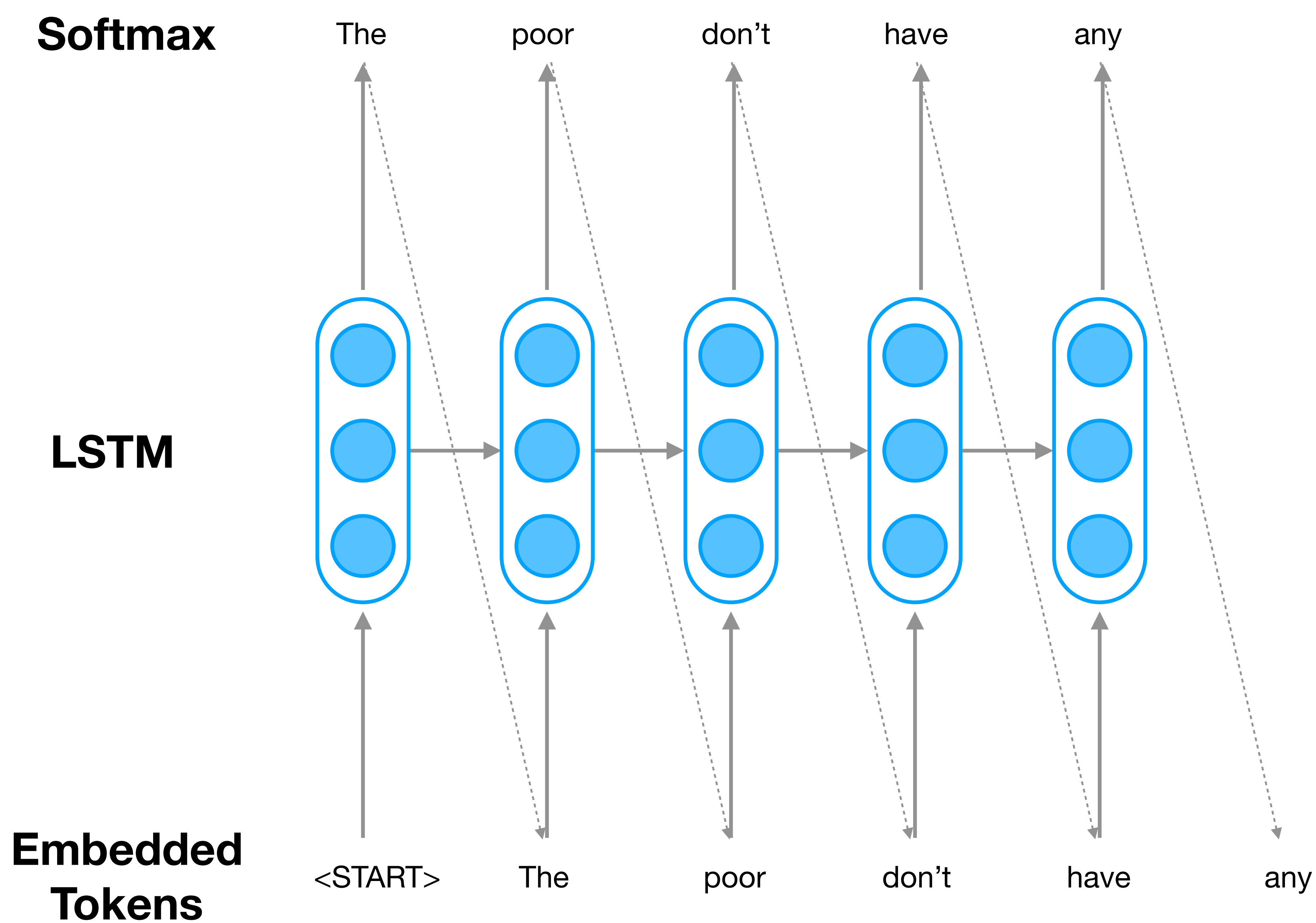
Language Model

Inference



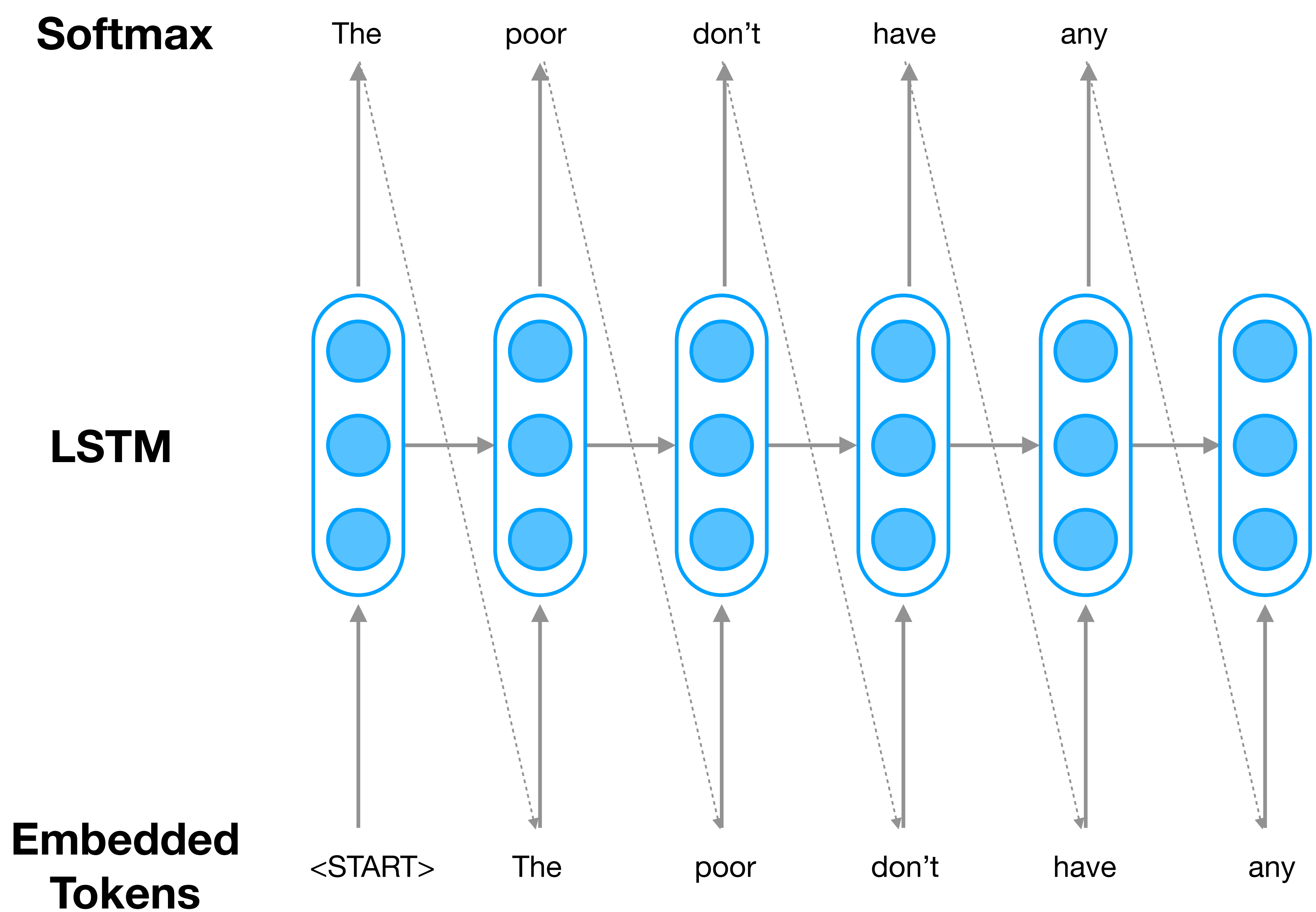
Language Model

Inference



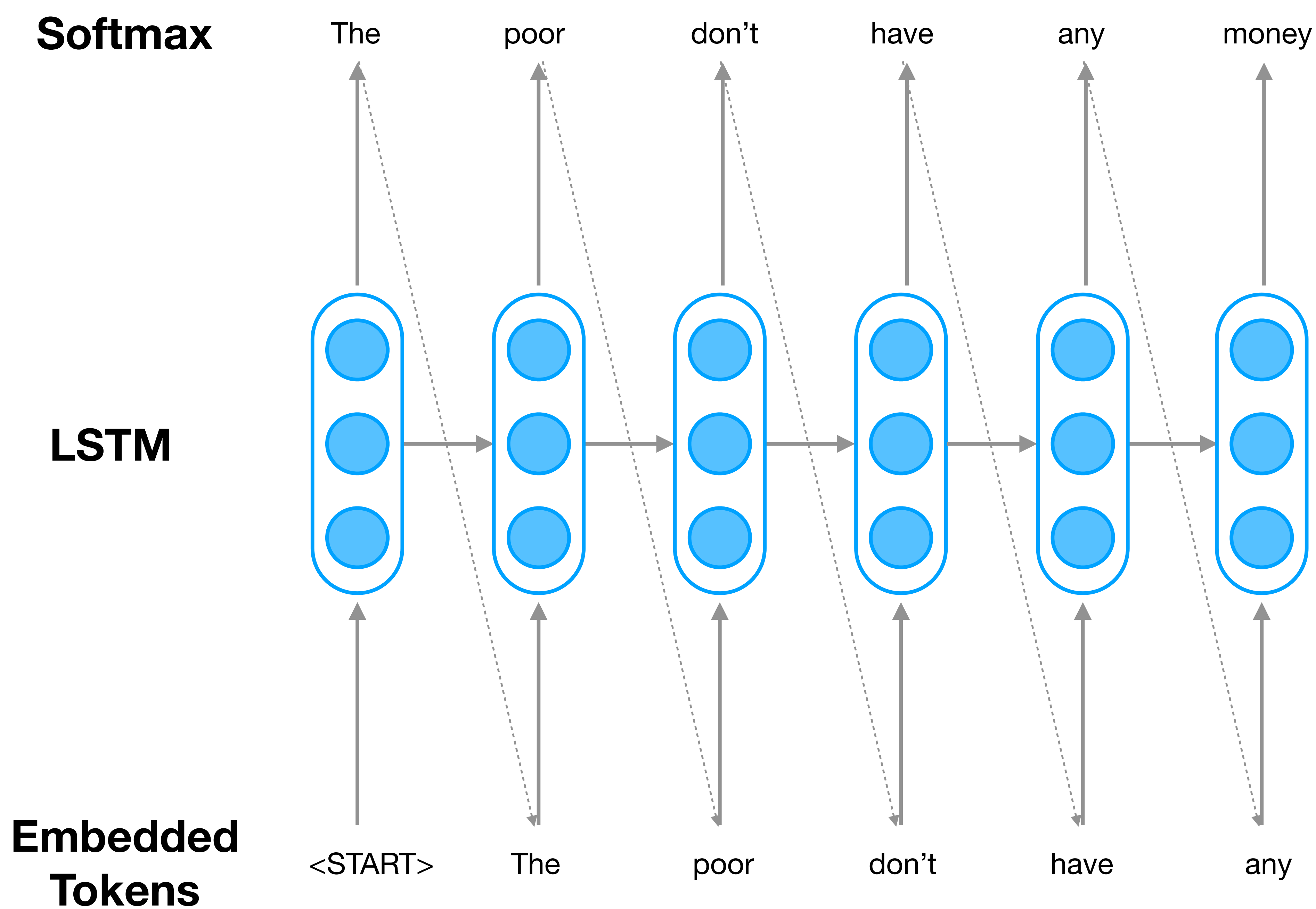
Language Model

Inference



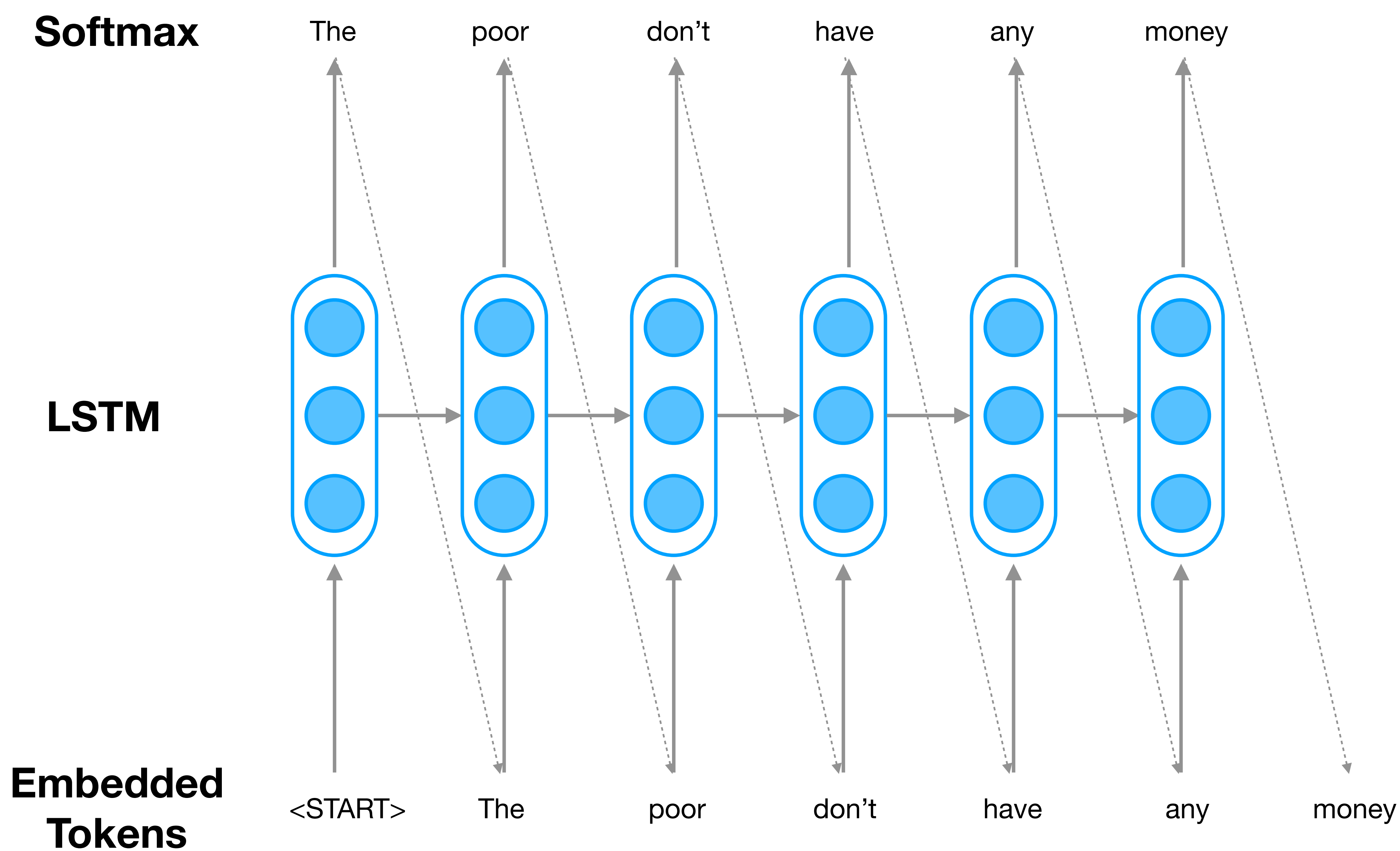
Language Model

Inference



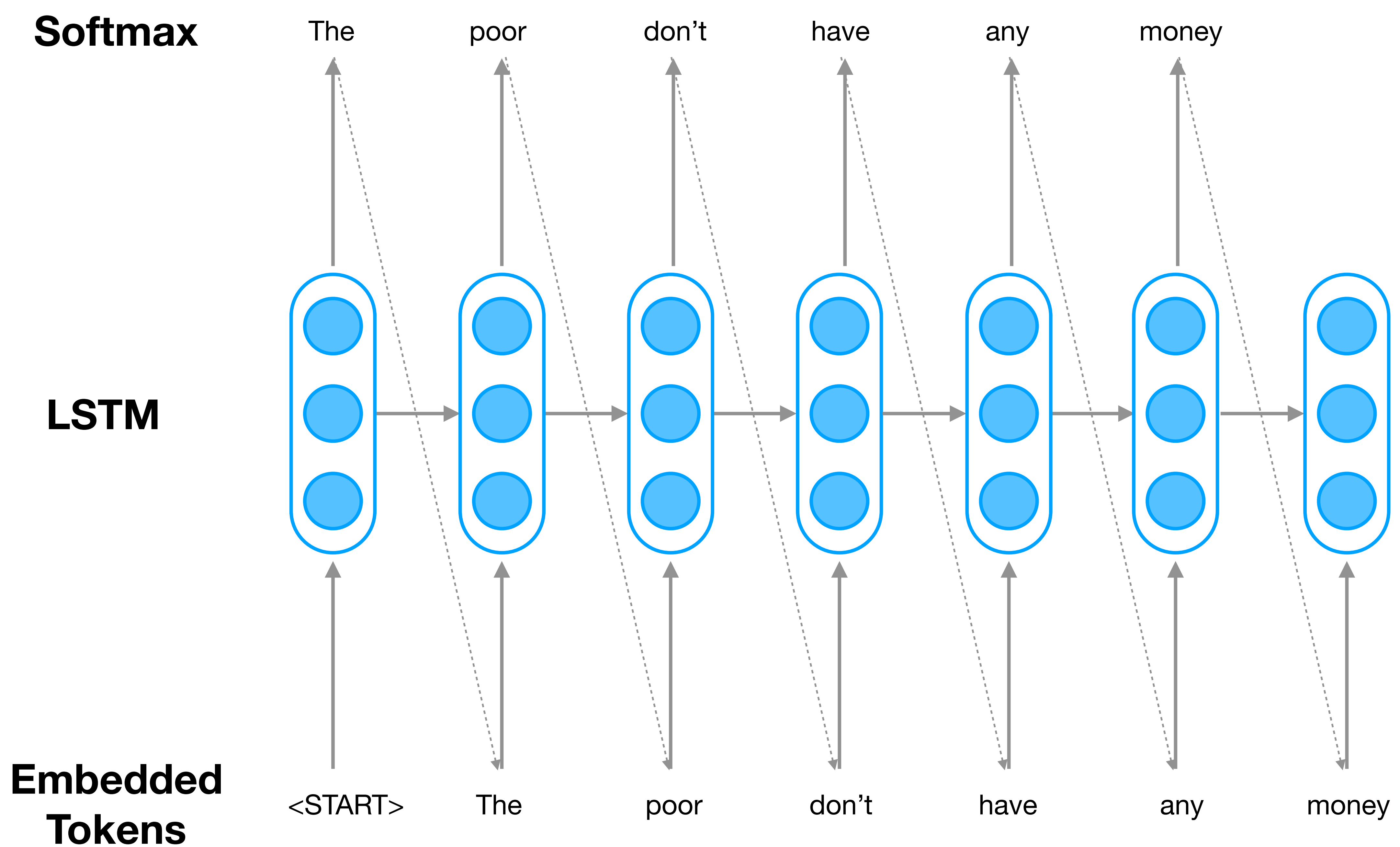
Language Model

Inference



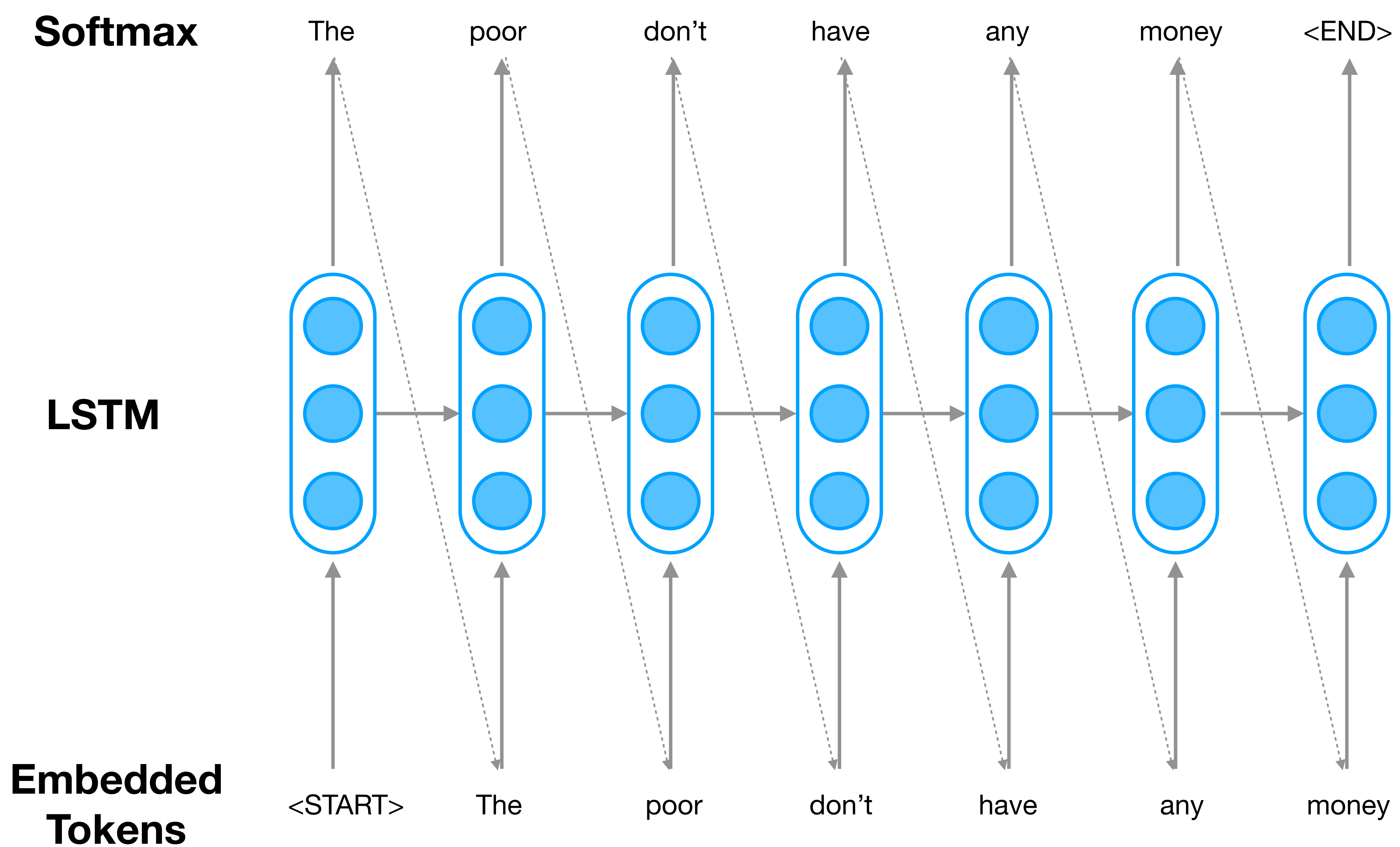
Language Model

Inference



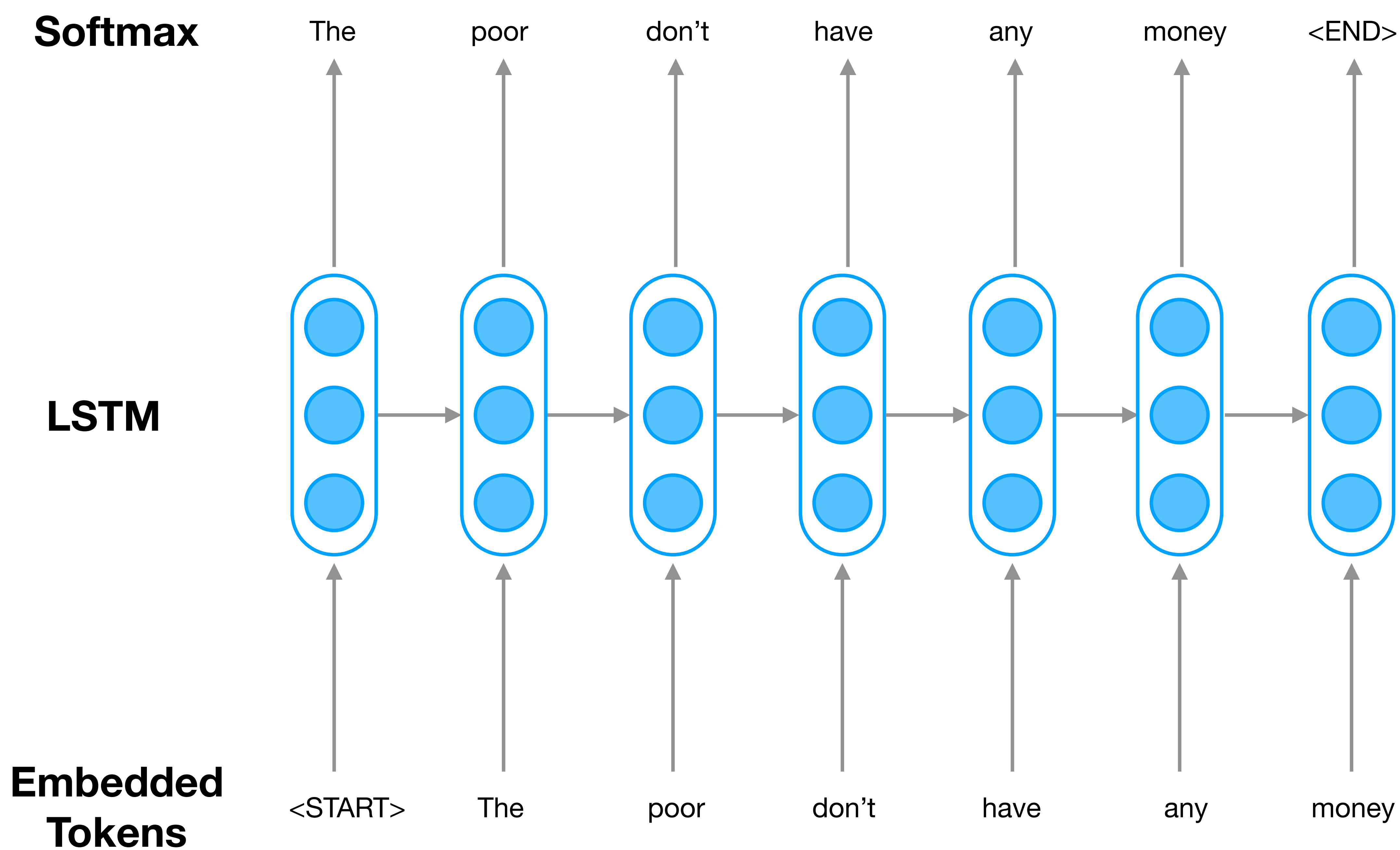
Language Model

Inference

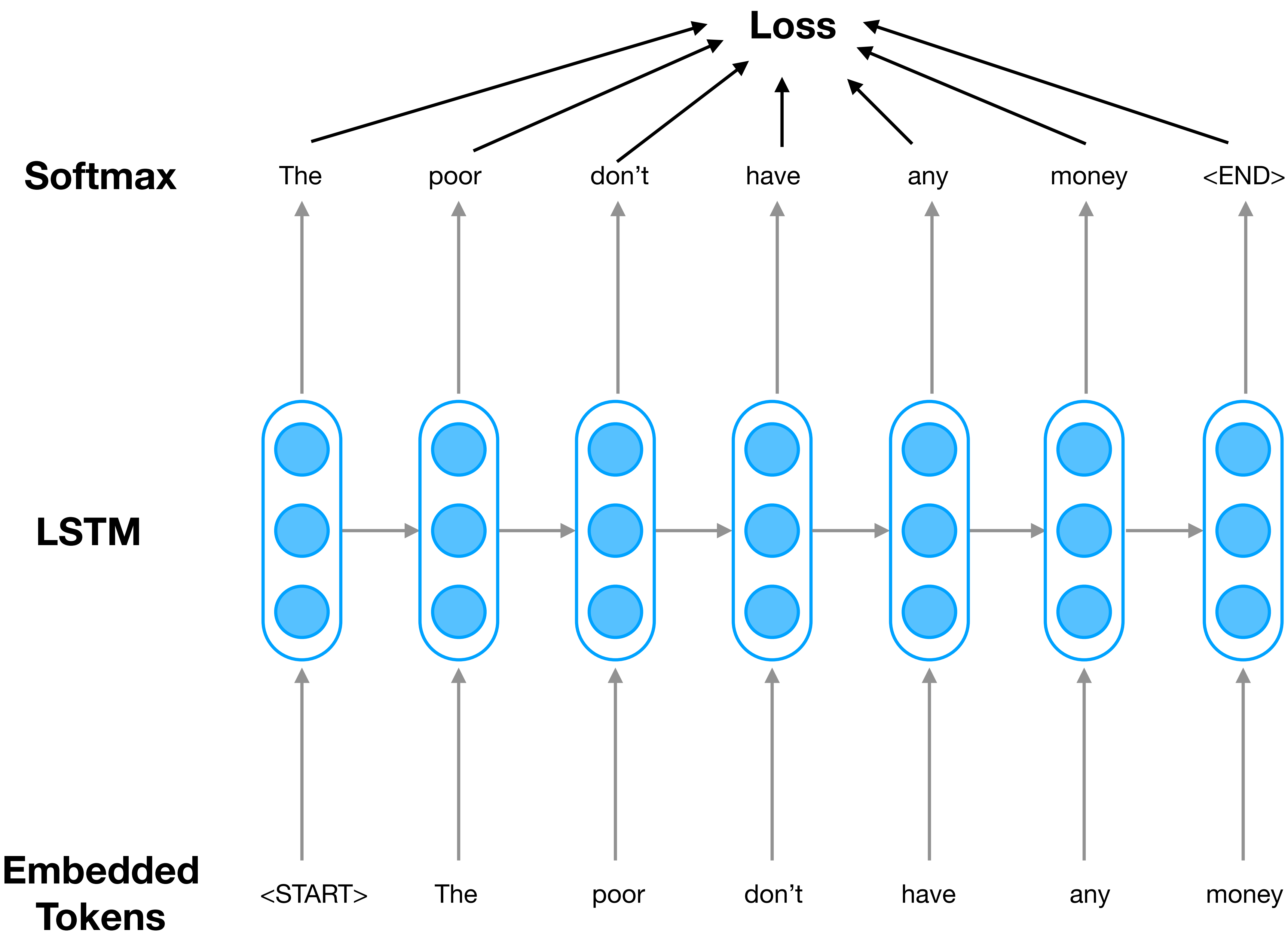


Language Model

Training

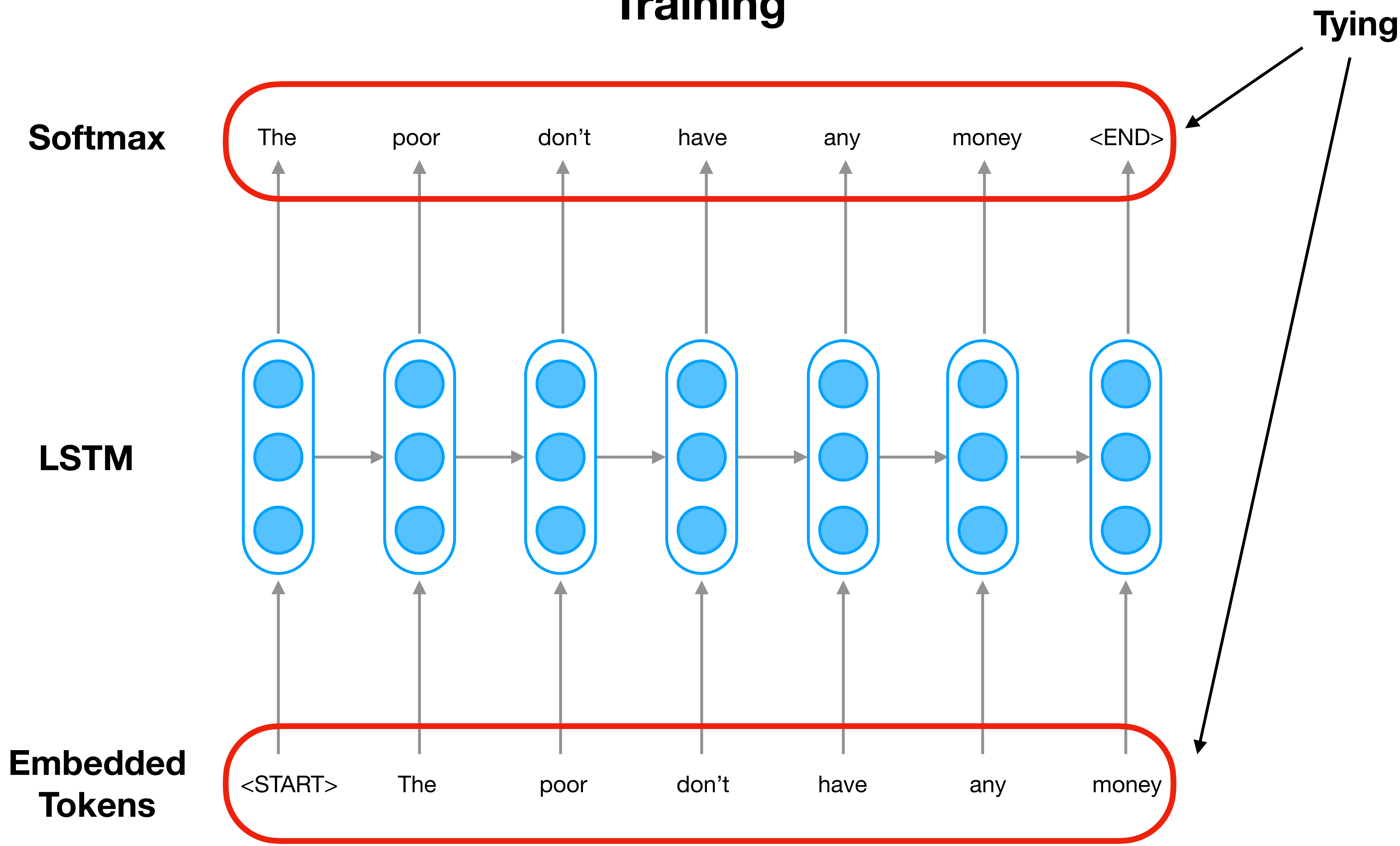


Language Model

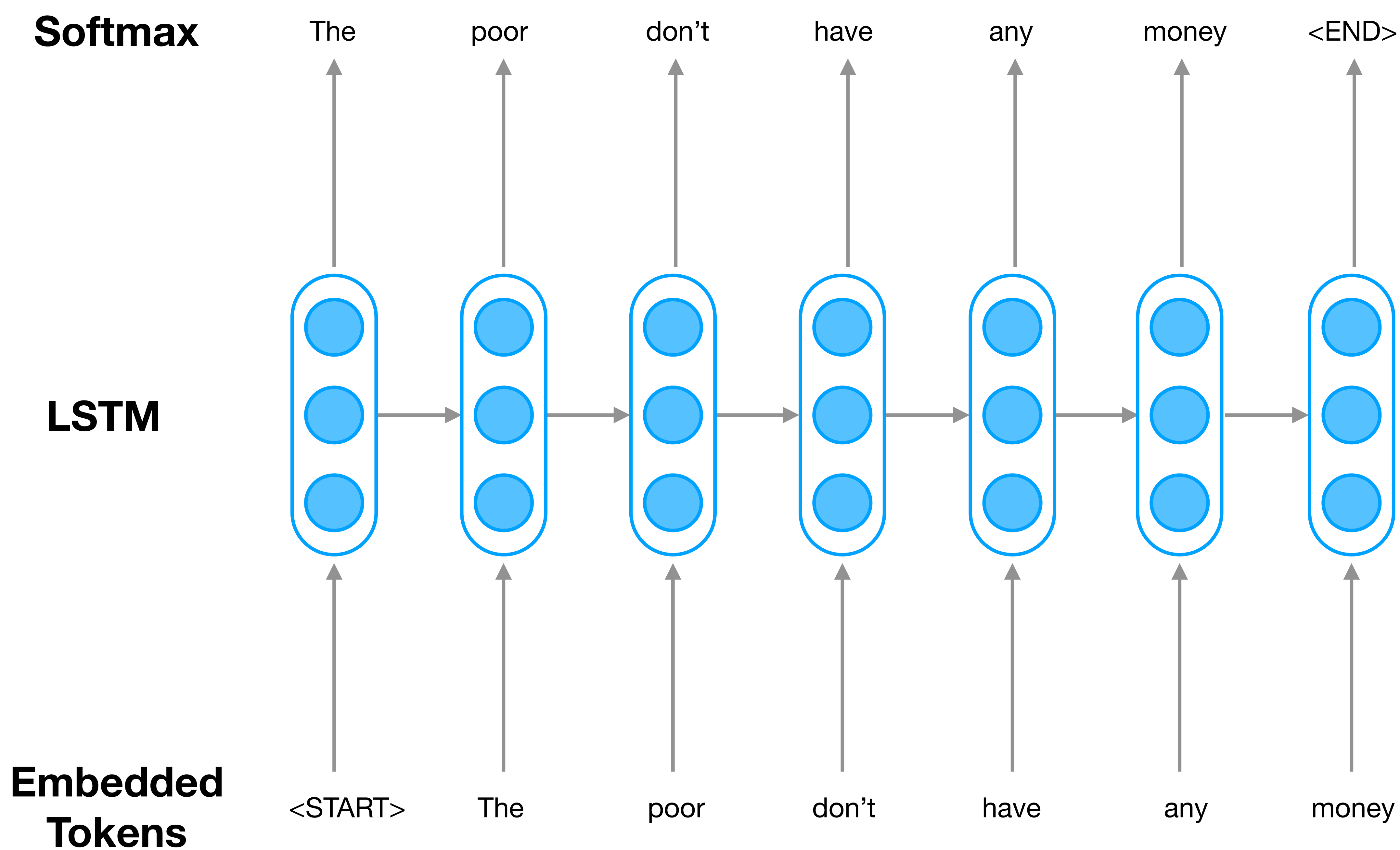


Language Model

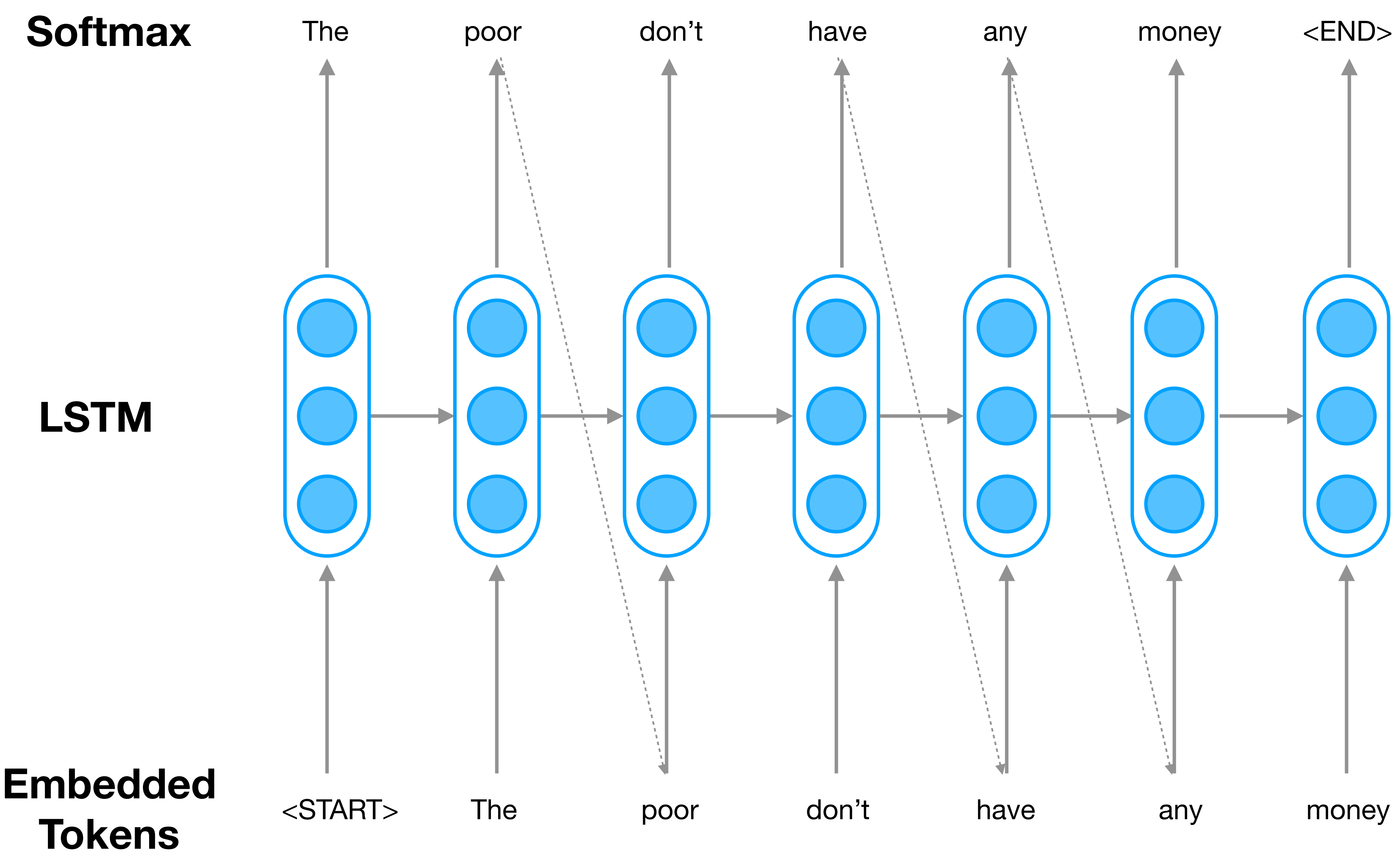
Training



Teacher Forcing



Teacher Forcing



Language Model

<START> The poor don’t have any money <END>

Language Model

X <START>

Y The

Language Model

X	<START>	The	
Y			poor

Language Model

X	<START>	The	poor	
Y				don’t

Language Model

X	<START>	The	poor	don’t	
Y					have

Language Model

X	<START>	The	poor	don’t	have	
Y						any

Language Model

X	<START>	The	poor	don't	have	any	
Y							money

Language Model

X	<START>	The	poor	don’t	have	any	money	
Y								<END>

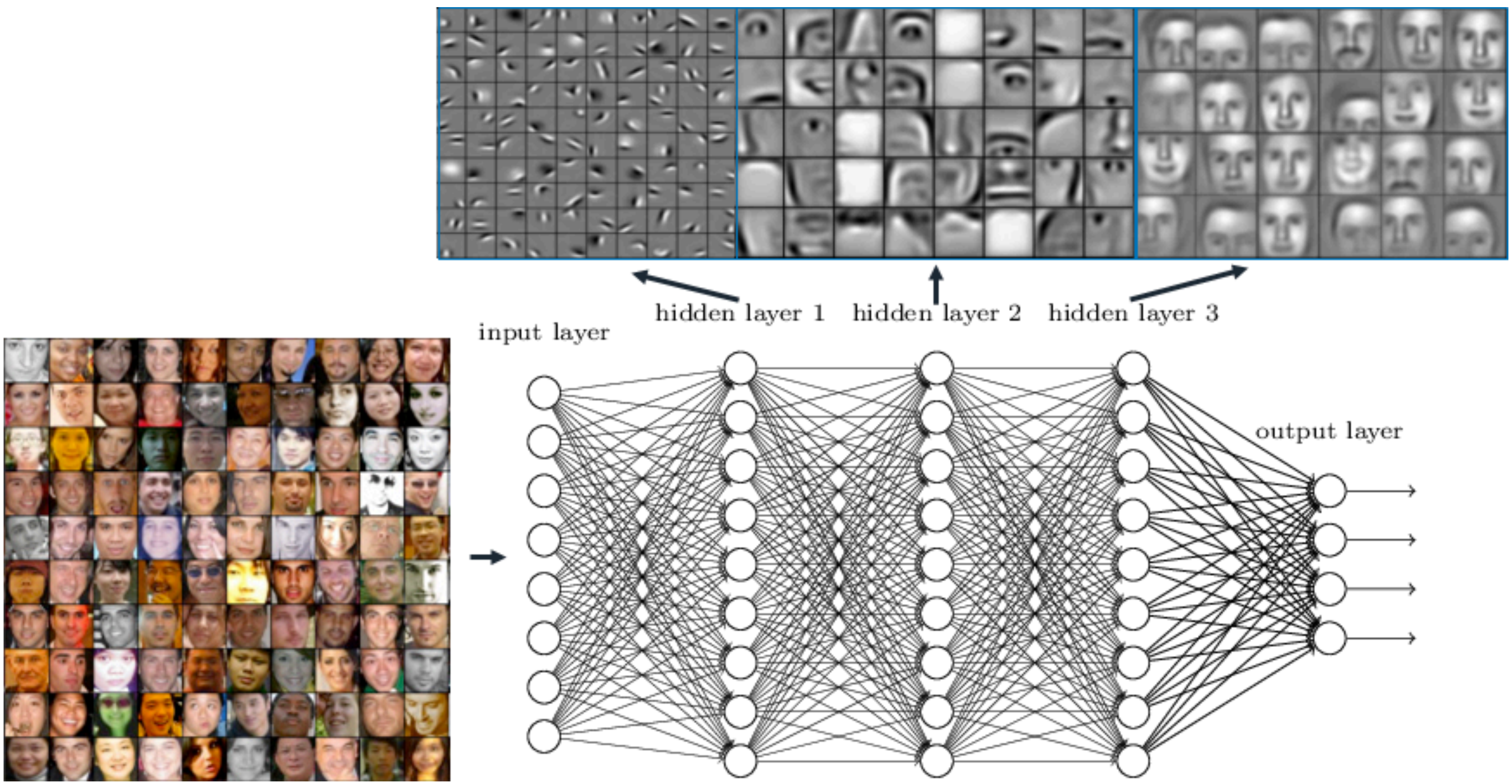
Language Model

X	<START>	The	poor	don’t	have	any	money	
Y								<END>

Sum of Losses

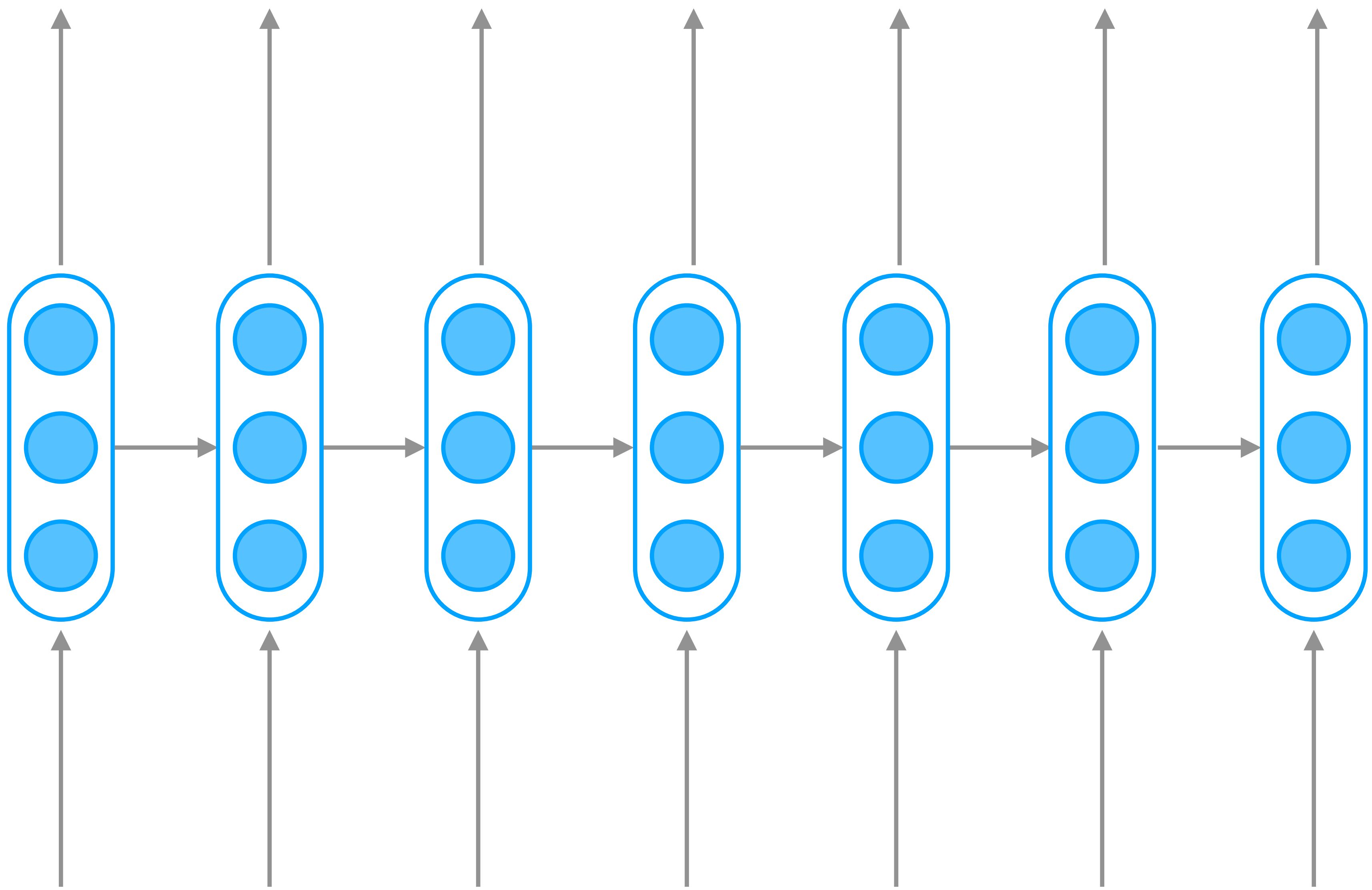
Transfer Learning

CNN Intuition



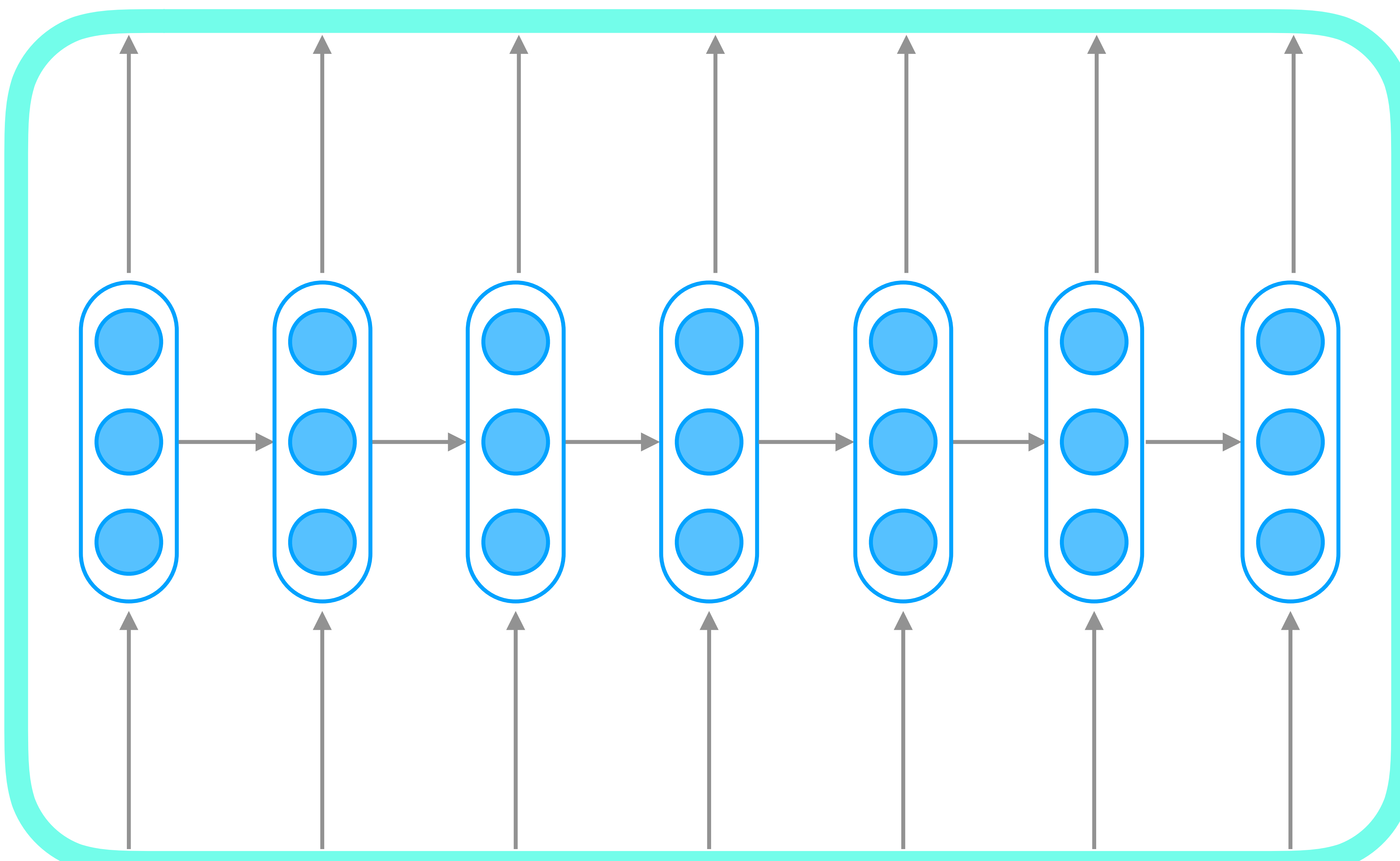
NLP LM Transfer Learning

LSTM

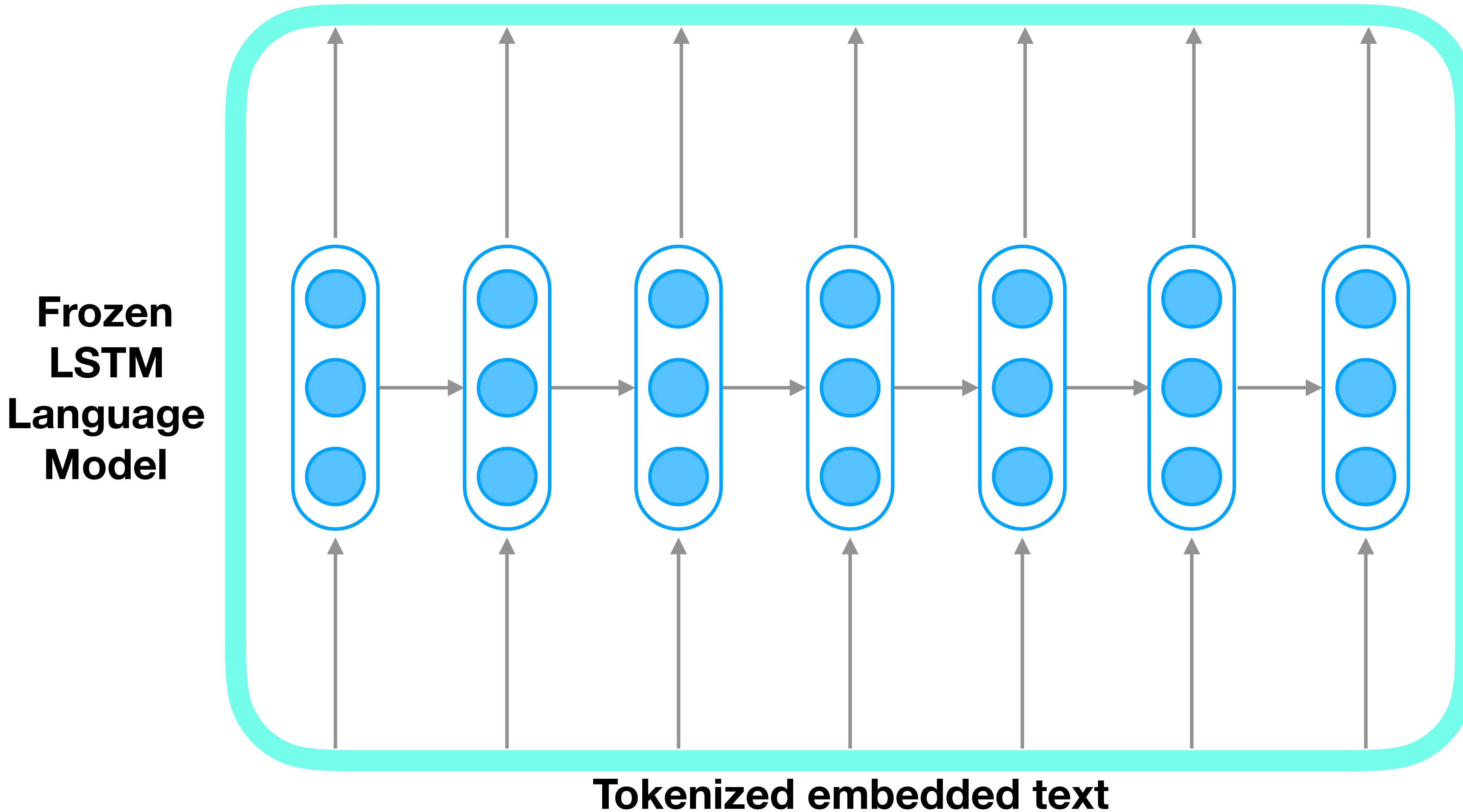


NLP LM Transfer Learning

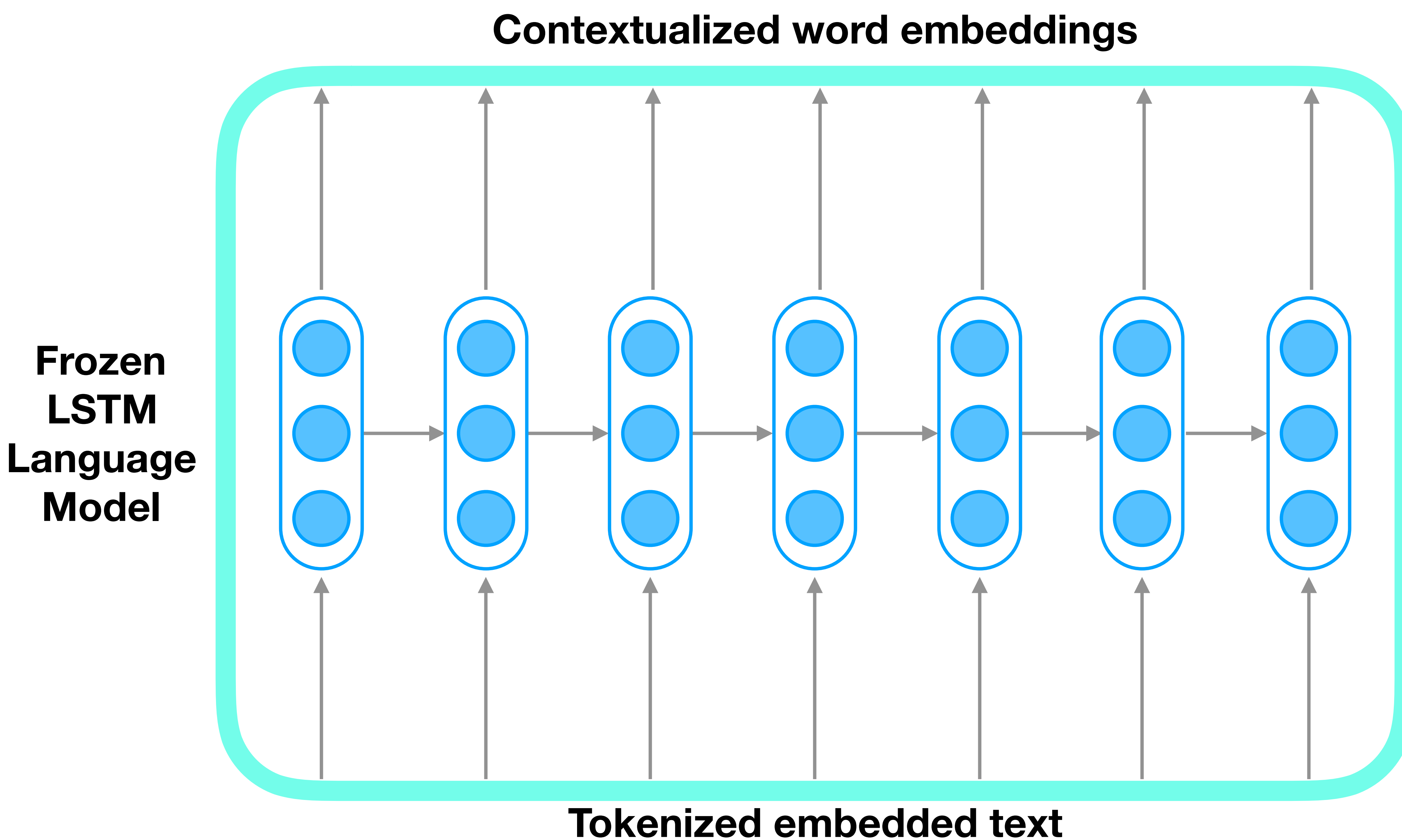
Frozen
LSTM
Language
Model



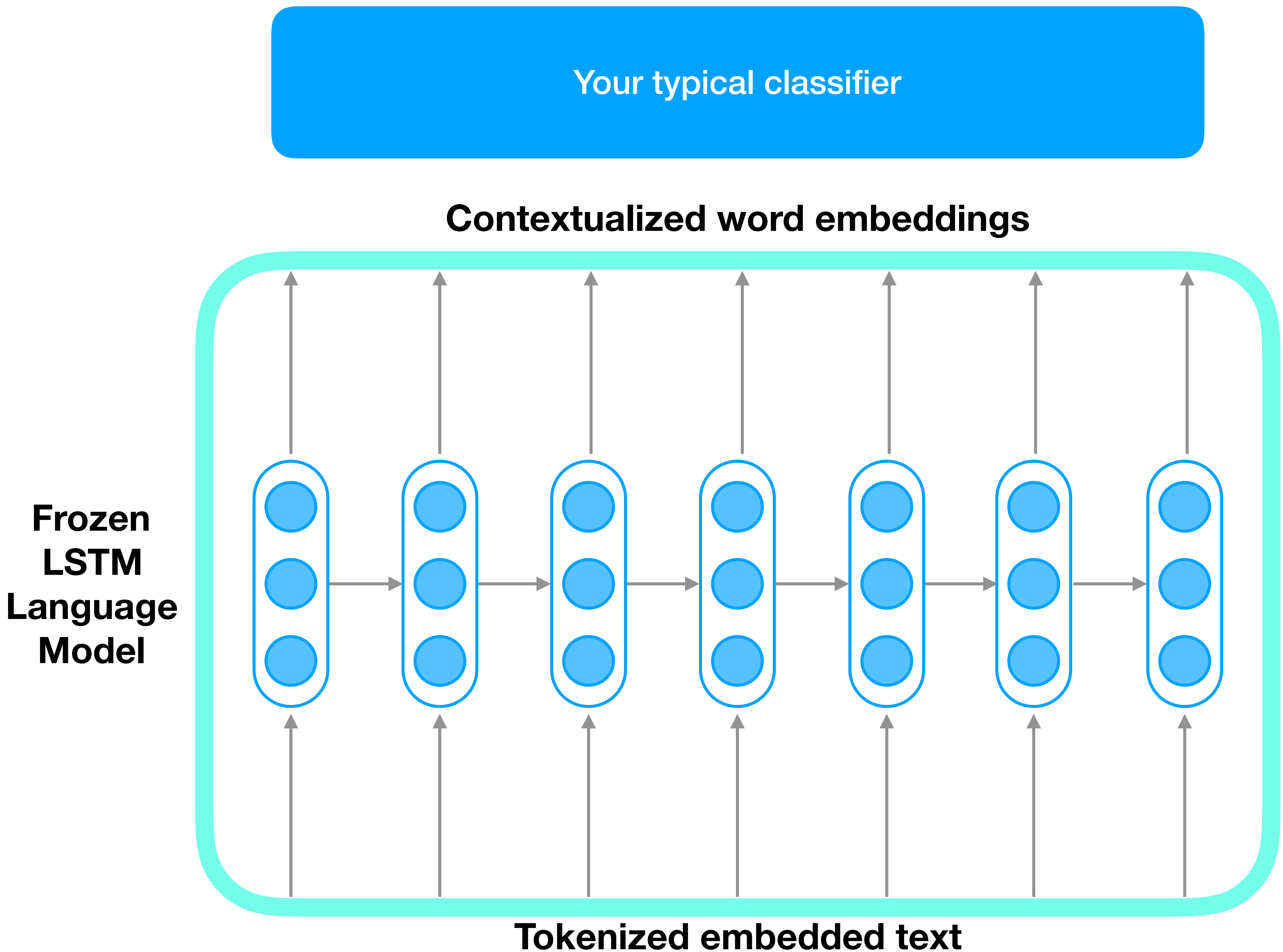
NLP LM Transfer Learning



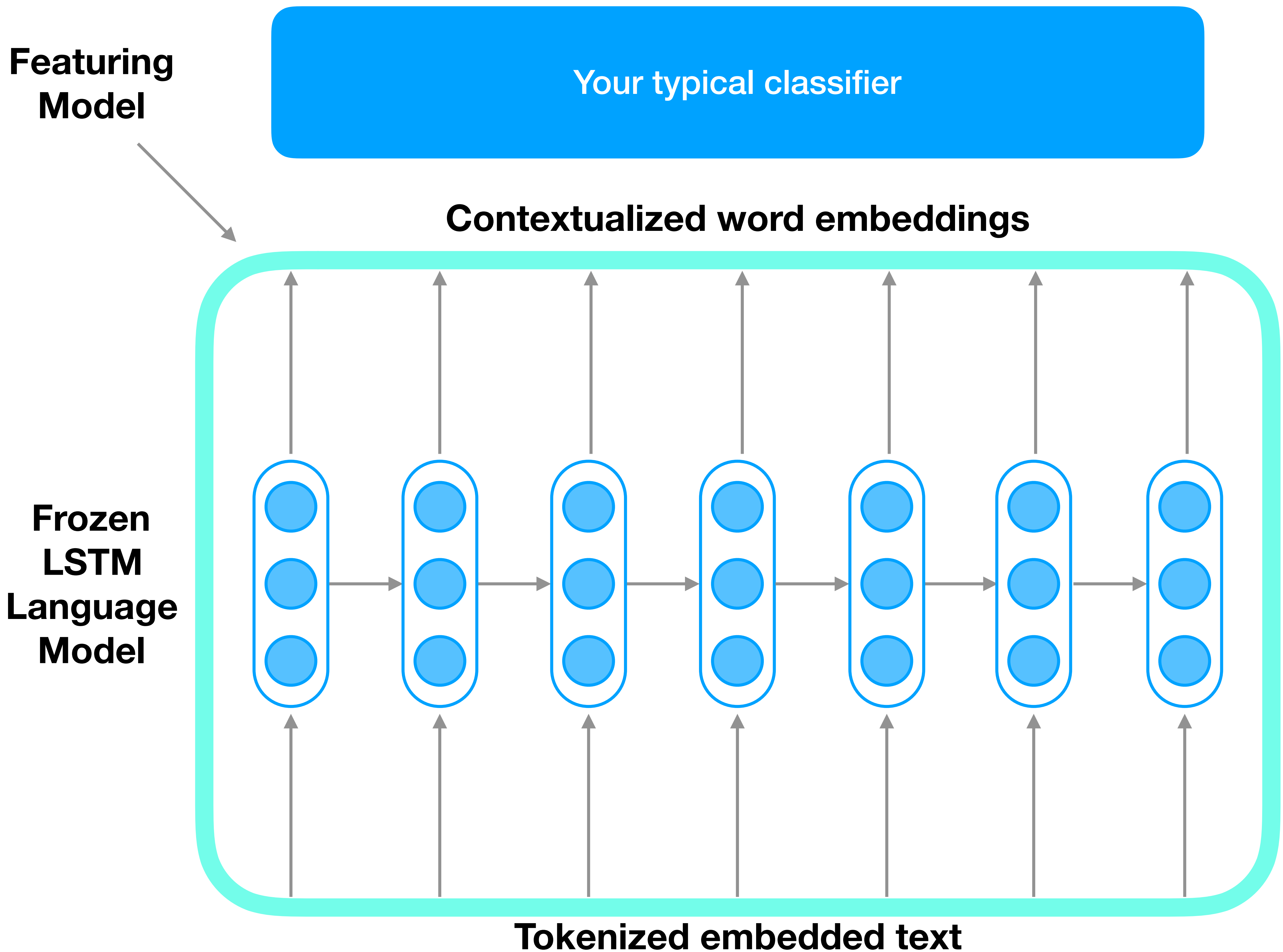
NLP LM Transfer Learning



NLP LM Transfer Learning

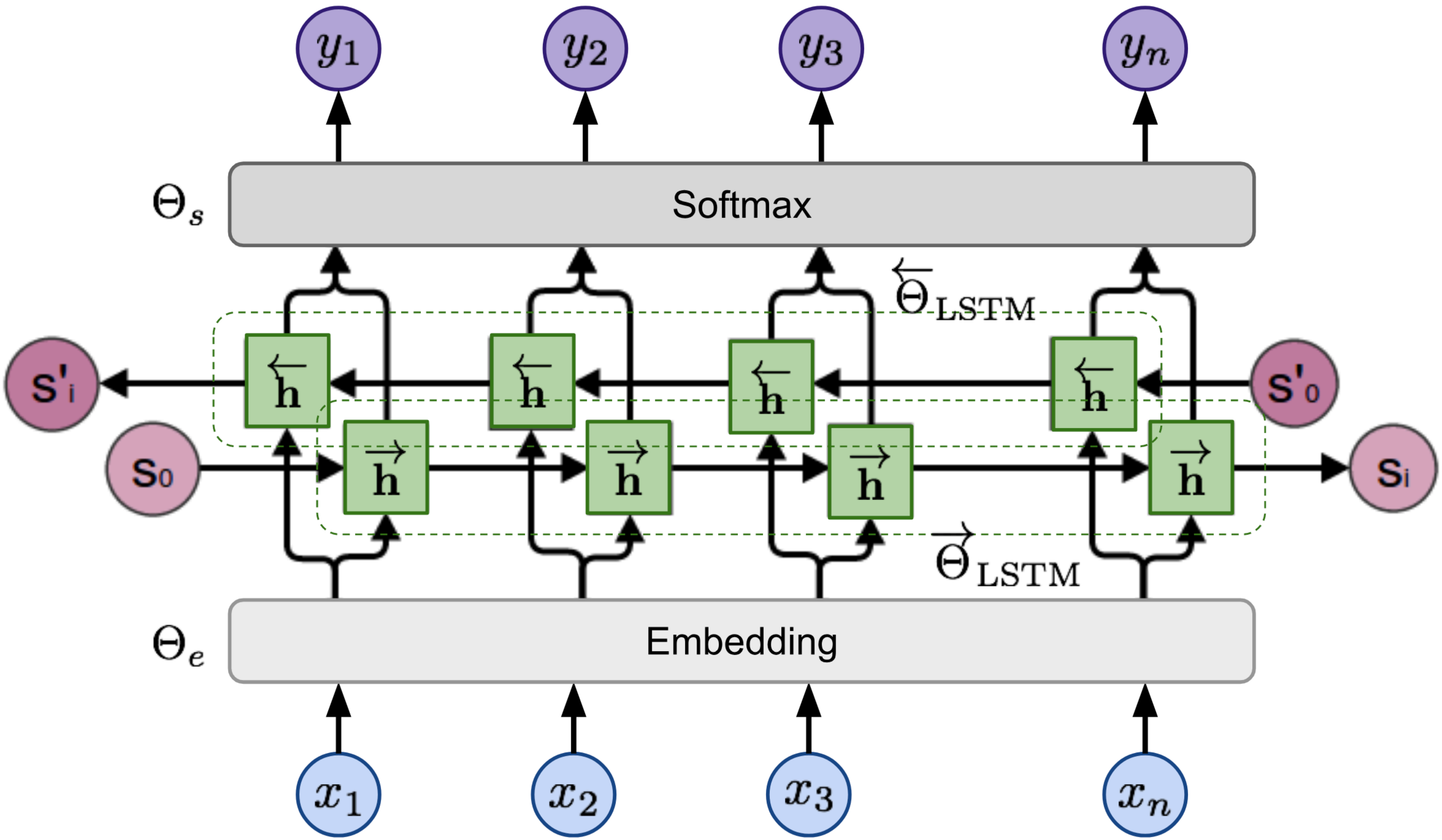


NLP LM Transfer Learning



ELMo

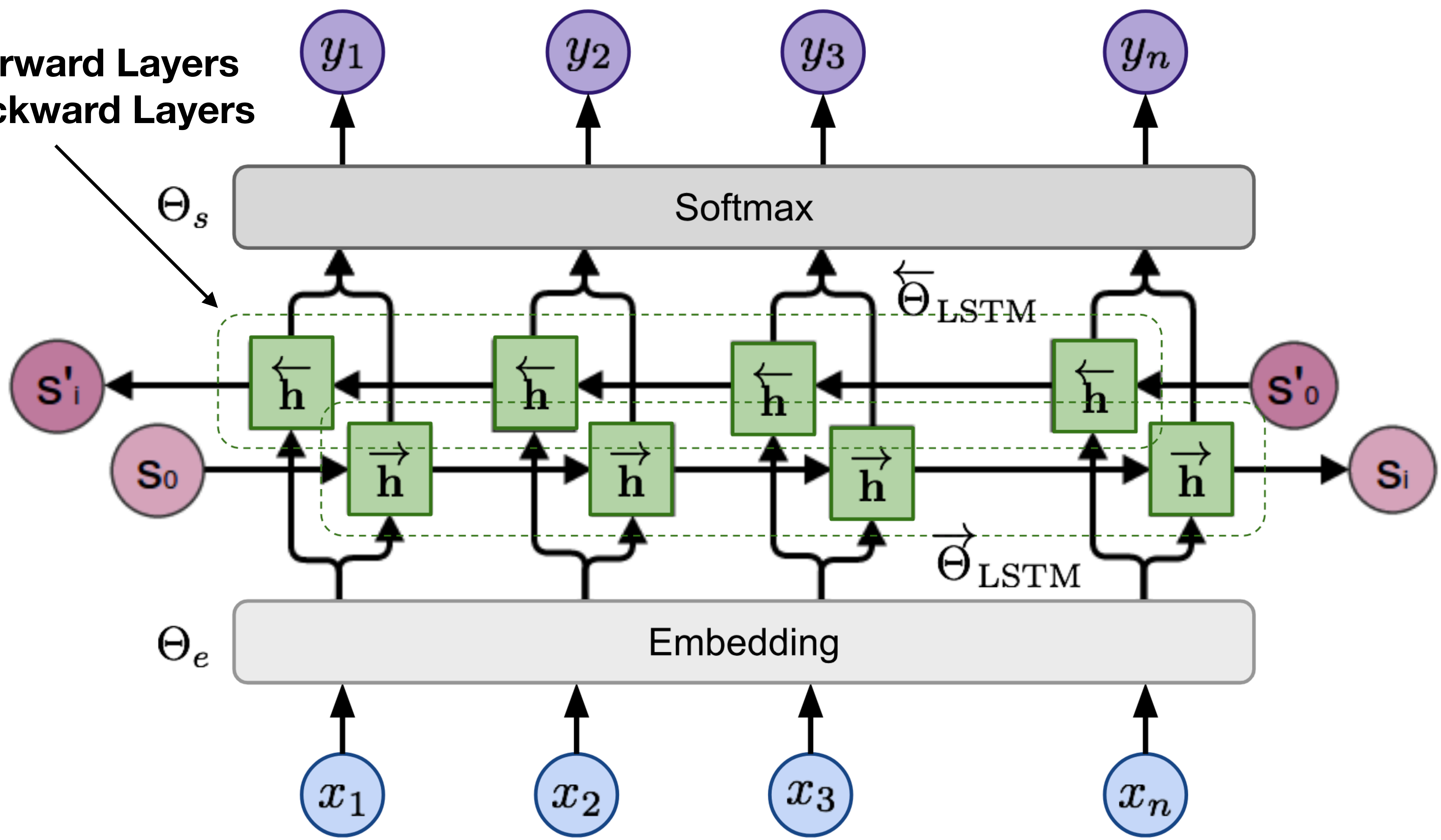
ELMo



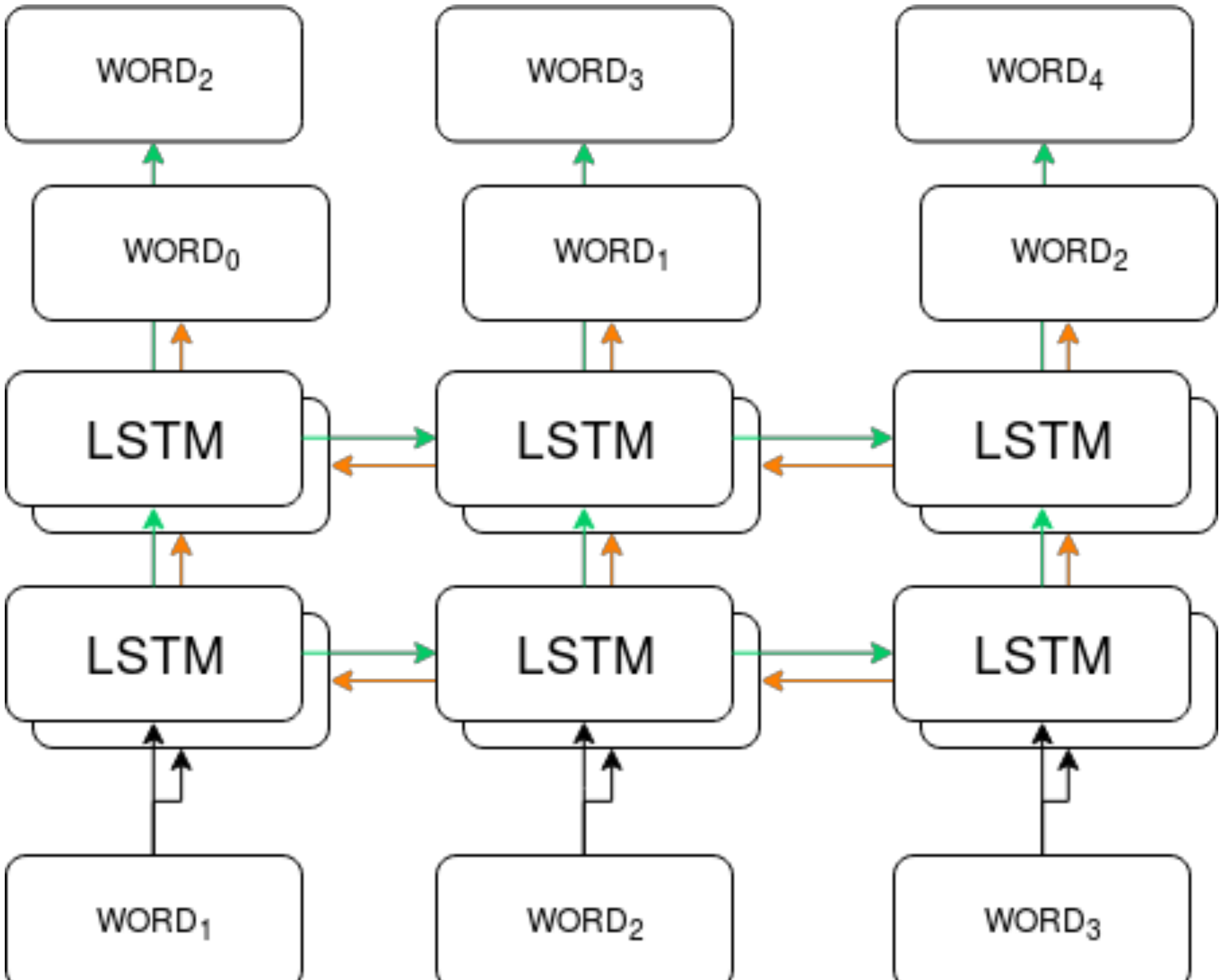
y_1
y_2
y_3
y_n
Θ_s
Softmax
$\overleftarrow{\Theta}_{\text{LSTM}}$
S'i
S'0
S0
Si
$\overleftarrow{\mathbf{h}}$
$\overrightarrow{\mathbf{h}}$
$\overrightarrow{\Theta}_{\text{LSTM}}$
Θ_e
Embedding
x_1
x_2
x_3
x_n

ELMo

2 Forward Layers
2 Backward Layers



ELMo



WORD2
WORD3
WORD4
WORD0
WORD1
WORD2
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
WORD1
WORD2
WORD3

ELMo

Source <START> The poor don’t have any money <END>

ELMo

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

ELMo

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

ELMo

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

ELMo

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

ELMo

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

ELMo

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

ELMo

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

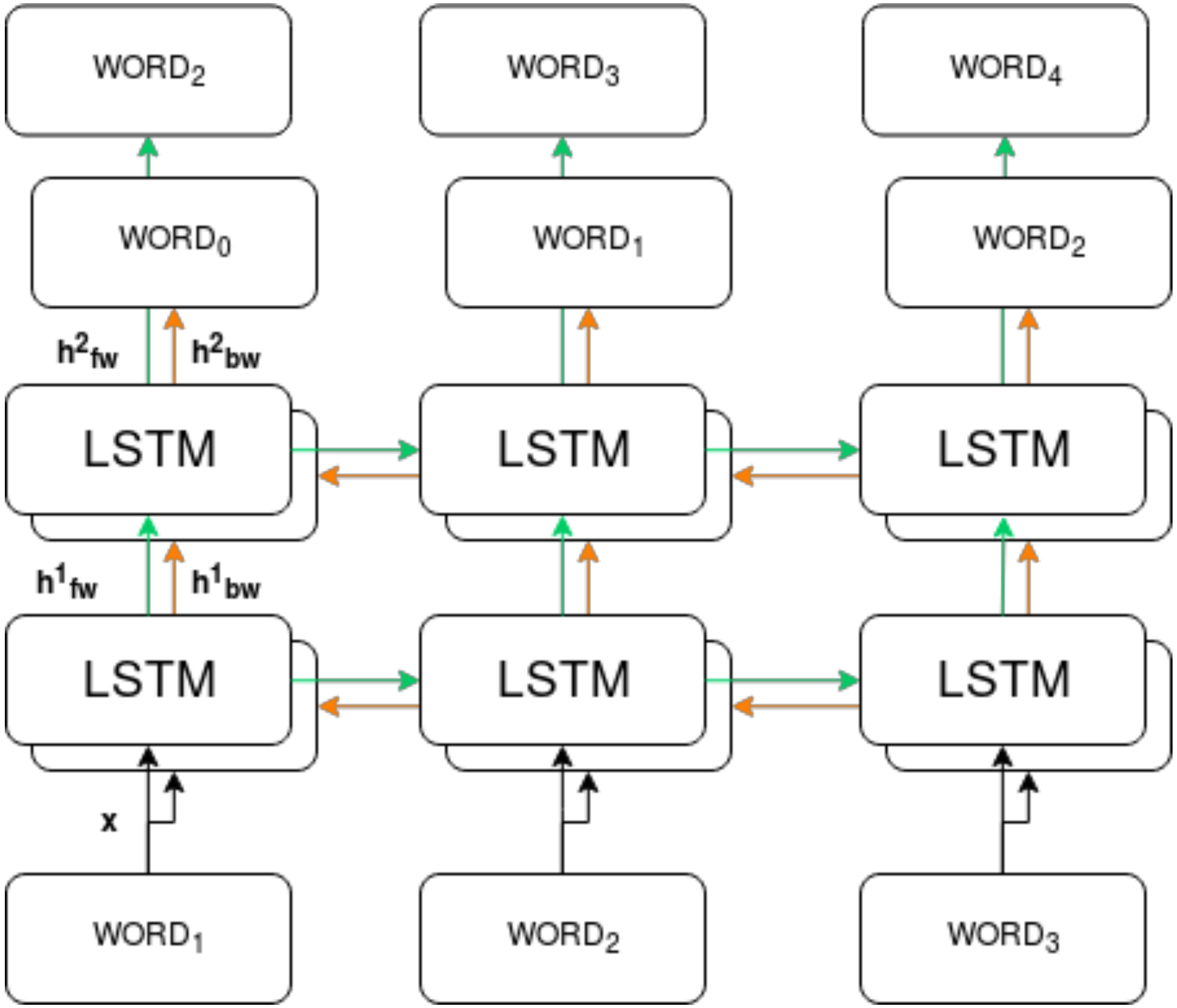
ELMo

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

ELMo

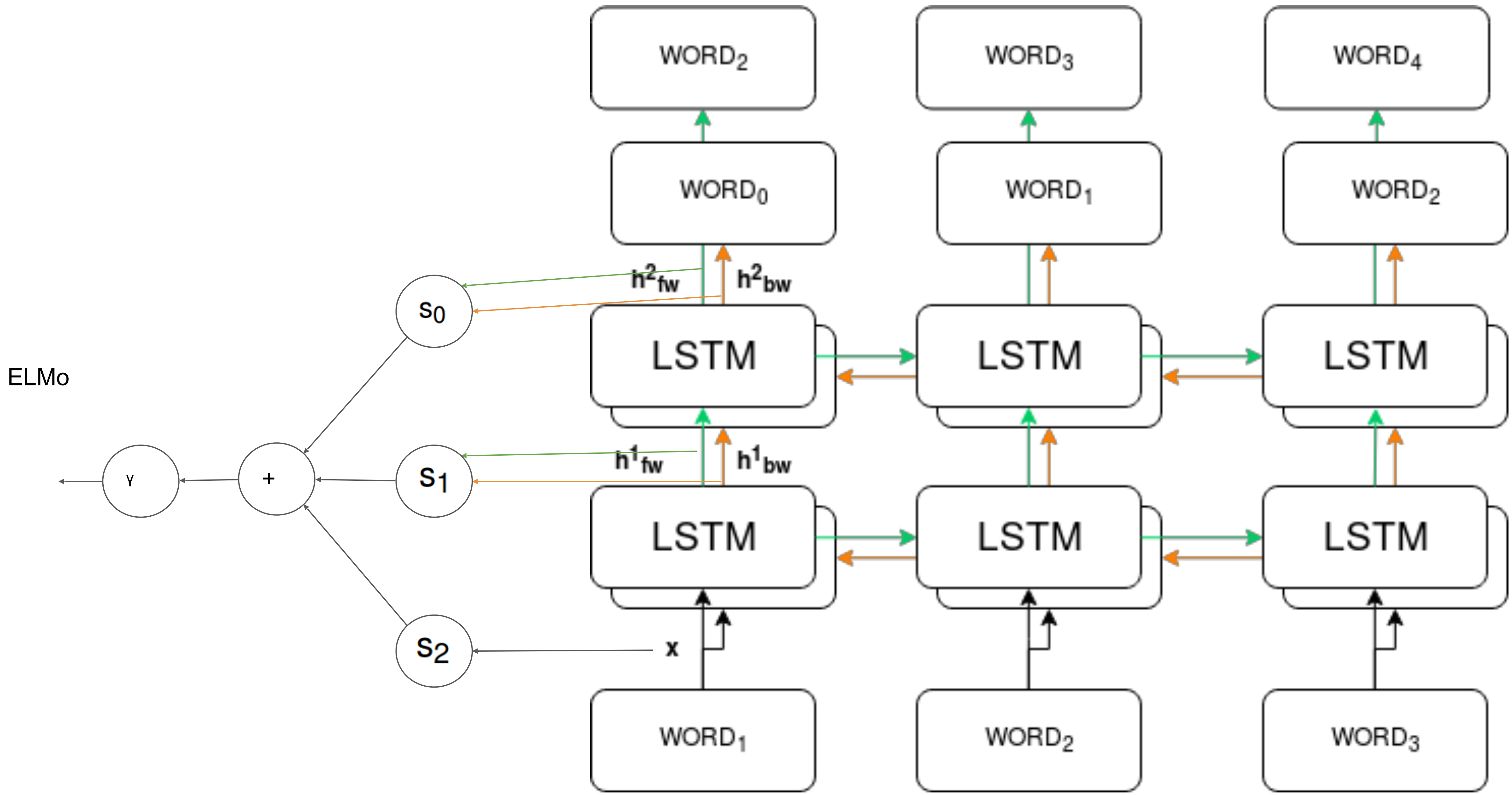
Source	<START>	The	poor	don’t	have	any	money	<END>
Forward	<START>	The	poor	don’t	have	any	money	<END>
Backward	<END>	money	any	have	don’t	poor	The	<START>

ELMo



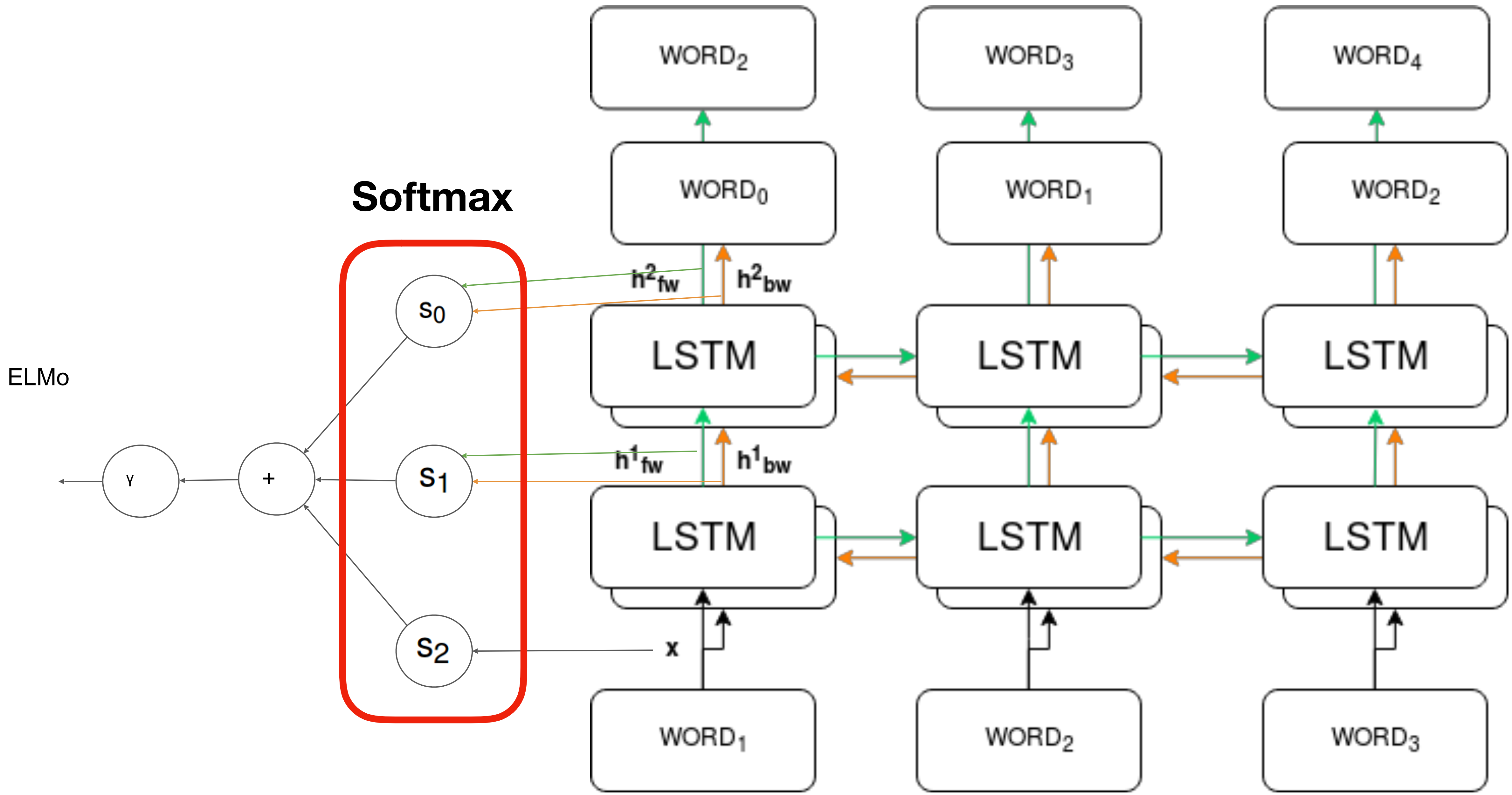
WORD2
WORD3
WORD4
WORD0
WORD1
WORD2
h2fw
h2bw
LSTM
LSTM
LSTM
h1fw
h1bw
LSTM
LSTM
LSTM
x
WORD1
WORD2
WORD3

ELMo



WORD$_2$
WORD$_3$
WORD$_4$
WORD$_0$
WORD$_1$
WORD$_2$
h^2_{fw}
h^2_{bw}
s_0
LSTM
LSTM
LSTM
ELMo
γ
+
s_1
h^1_{fw}
h^1_{bw}
LSTM
LSTM
LSTM
s_2
x
WORD$_1$
WORD$_2$
WORD$_3$

ELMo



Softmax
ELMo
γ
+
s_0
s_1
s_2
h^2_{fw}
h^2_{bw}
h^1_{fw}
h^1_{bw}
x
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

Results

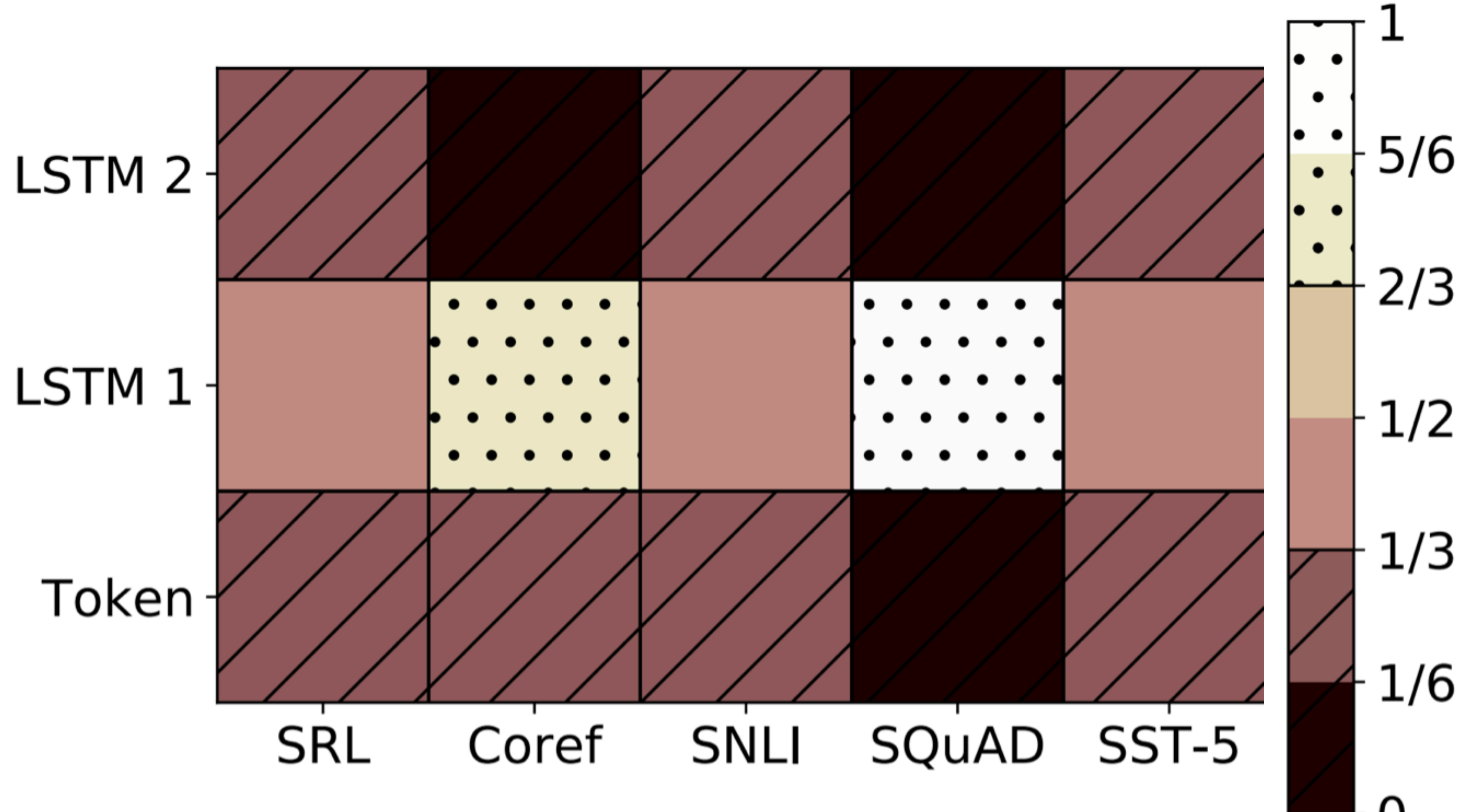
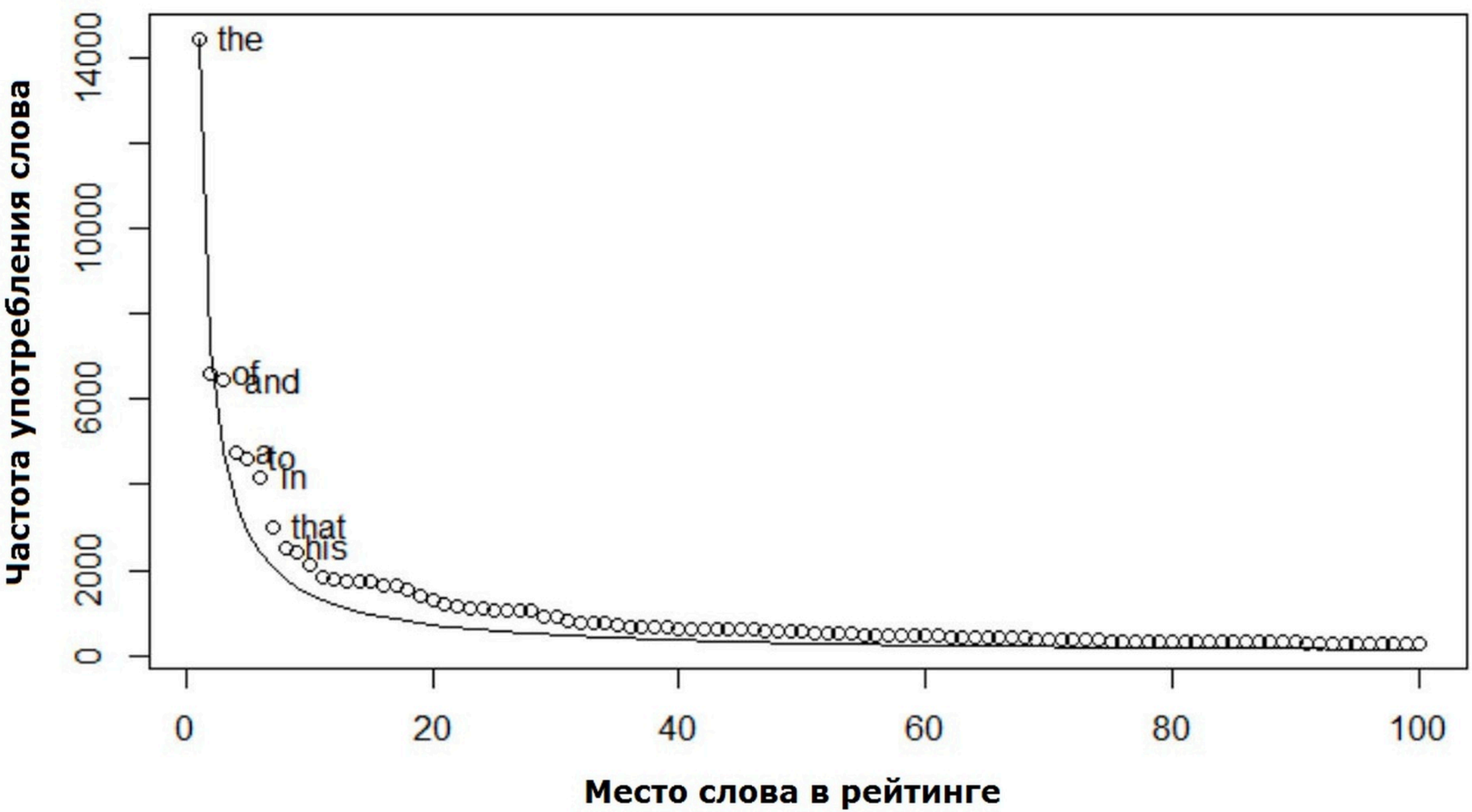


Figure 2: Visualization of softmax normalized biLM layer weights across tasks and ELMo locations. Normalized weights less then $1/3$ are hatched with horizontal lines and those greater then $2/3$ are speckled.

Tokenization

Закон Ципфа



Tokenization

Word level		**Character level**	
+	Small text length	-	Long text
-	Big vocabulary size	+	Small vocabulary size
-	OOV	+	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']

https://vk.com/deepvk?z=video-133150806_456239021%2Fvideos-166490678

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']

https://vk.com/deepvk?z=video-133150806_456239021%2Fvideos-166490678

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'}.

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>’

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

Sequence to sequence

Source sentence

Sequence to sequence

Les pauvres sont démunis

Source sentence

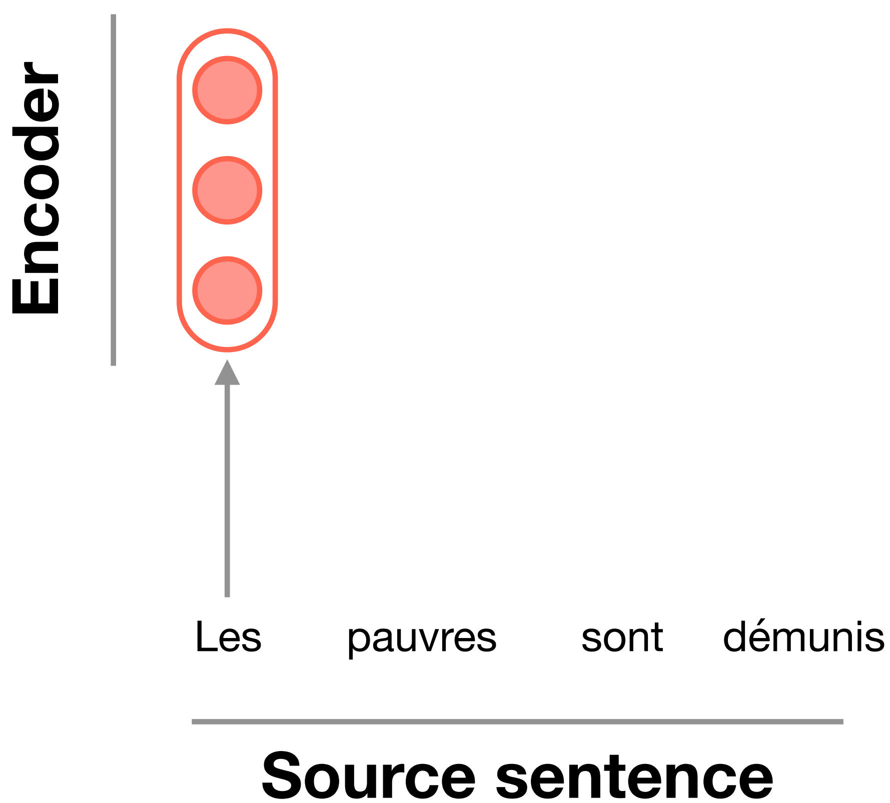
Sequence to sequence

Encoder

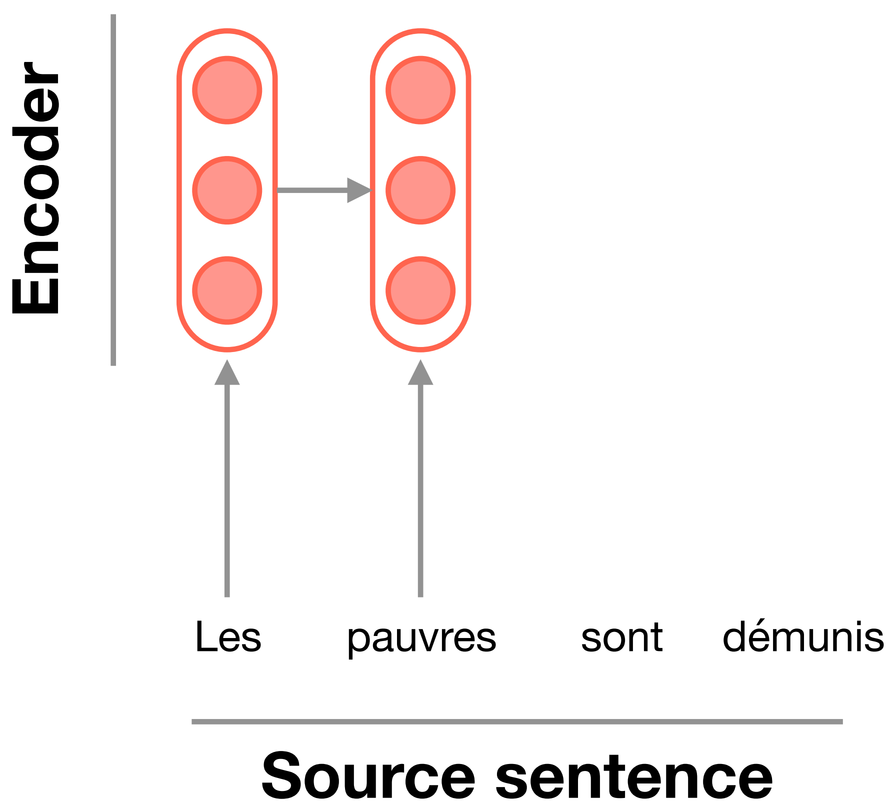
Les pauvres sont démunis

Source sentence

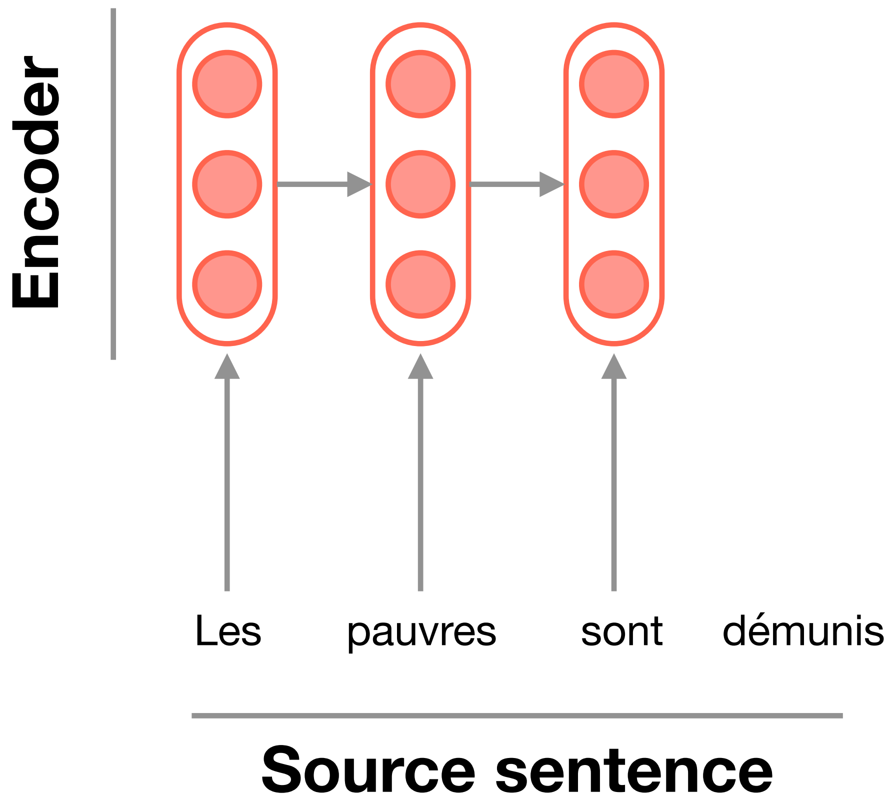
Sequence to sequence



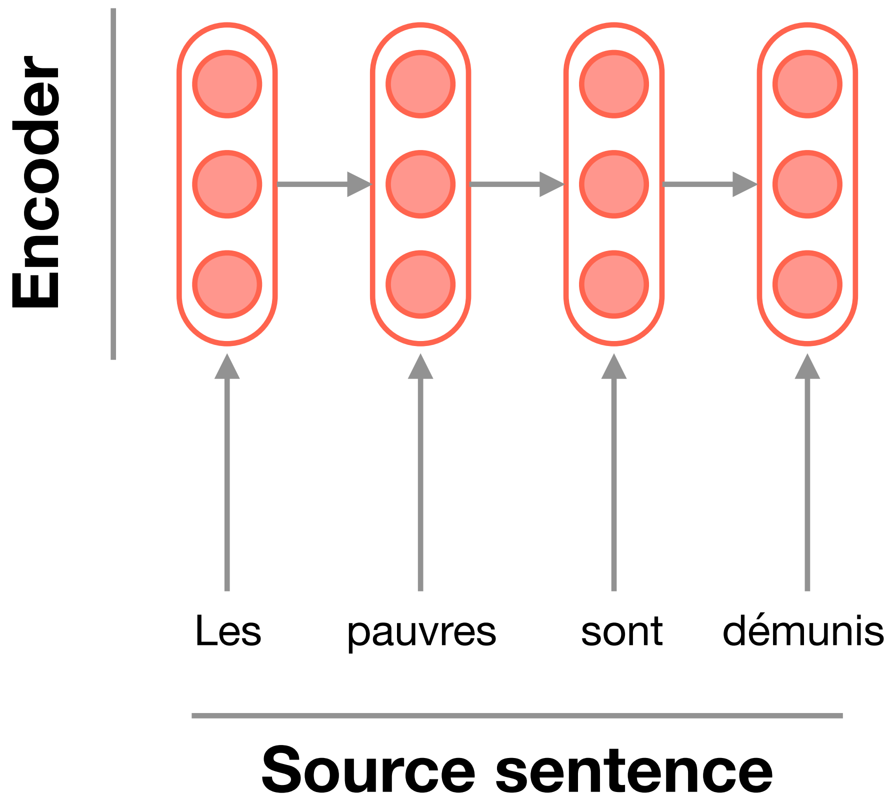
Sequence to sequence



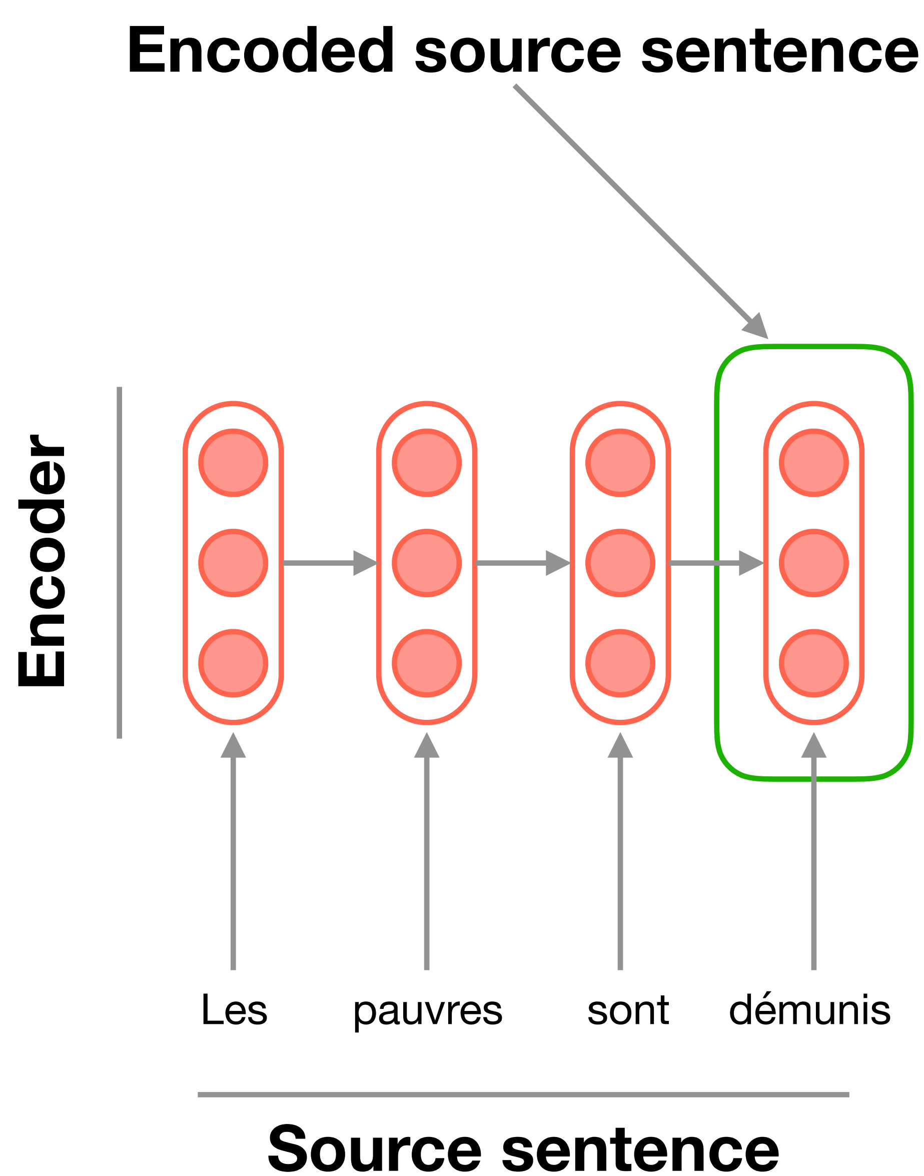
Sequence to sequence



Sequence to sequence

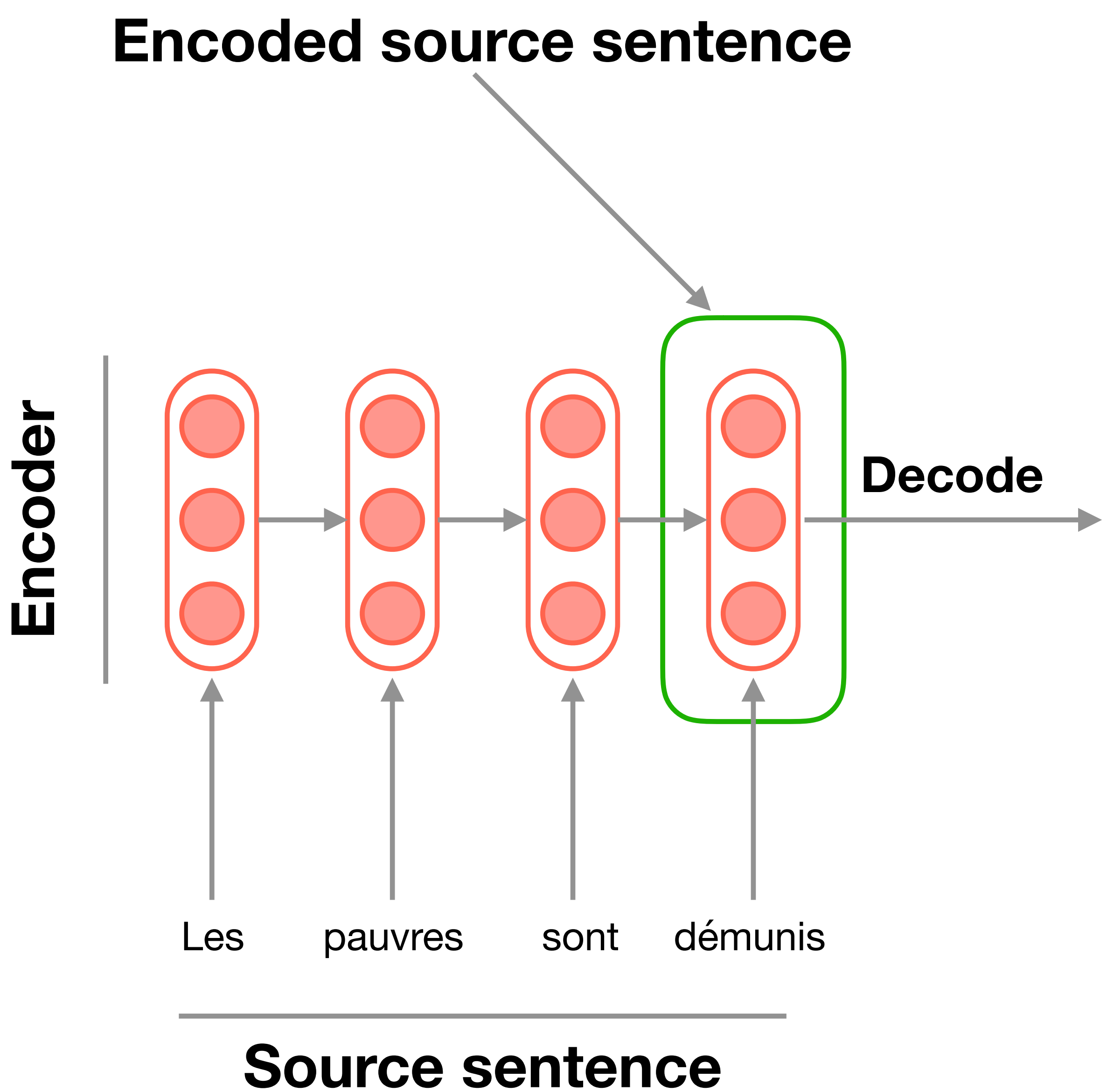


Sequence to sequence



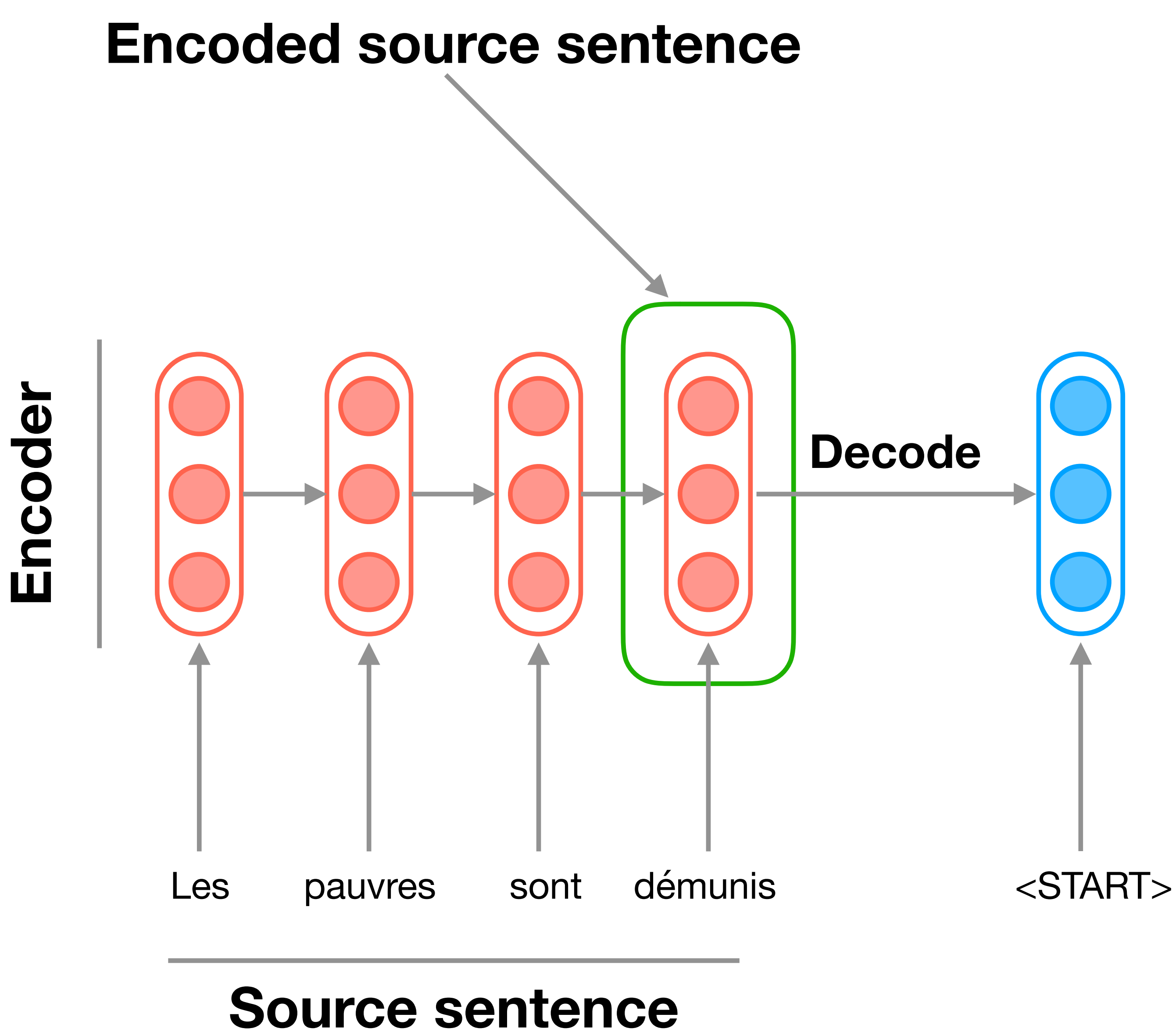
Sequence to sequence

Inference



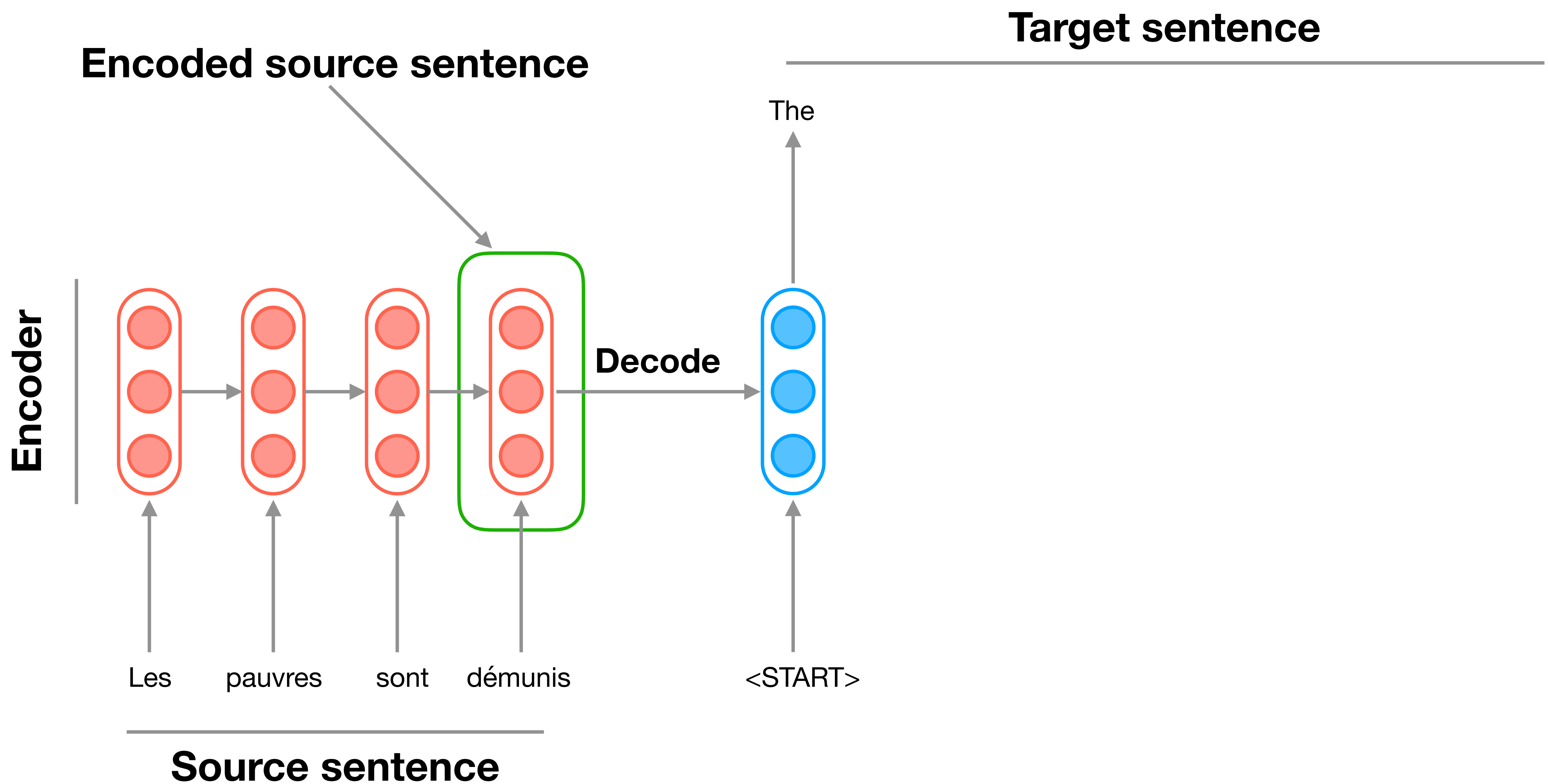
Sequence to sequence

Inference



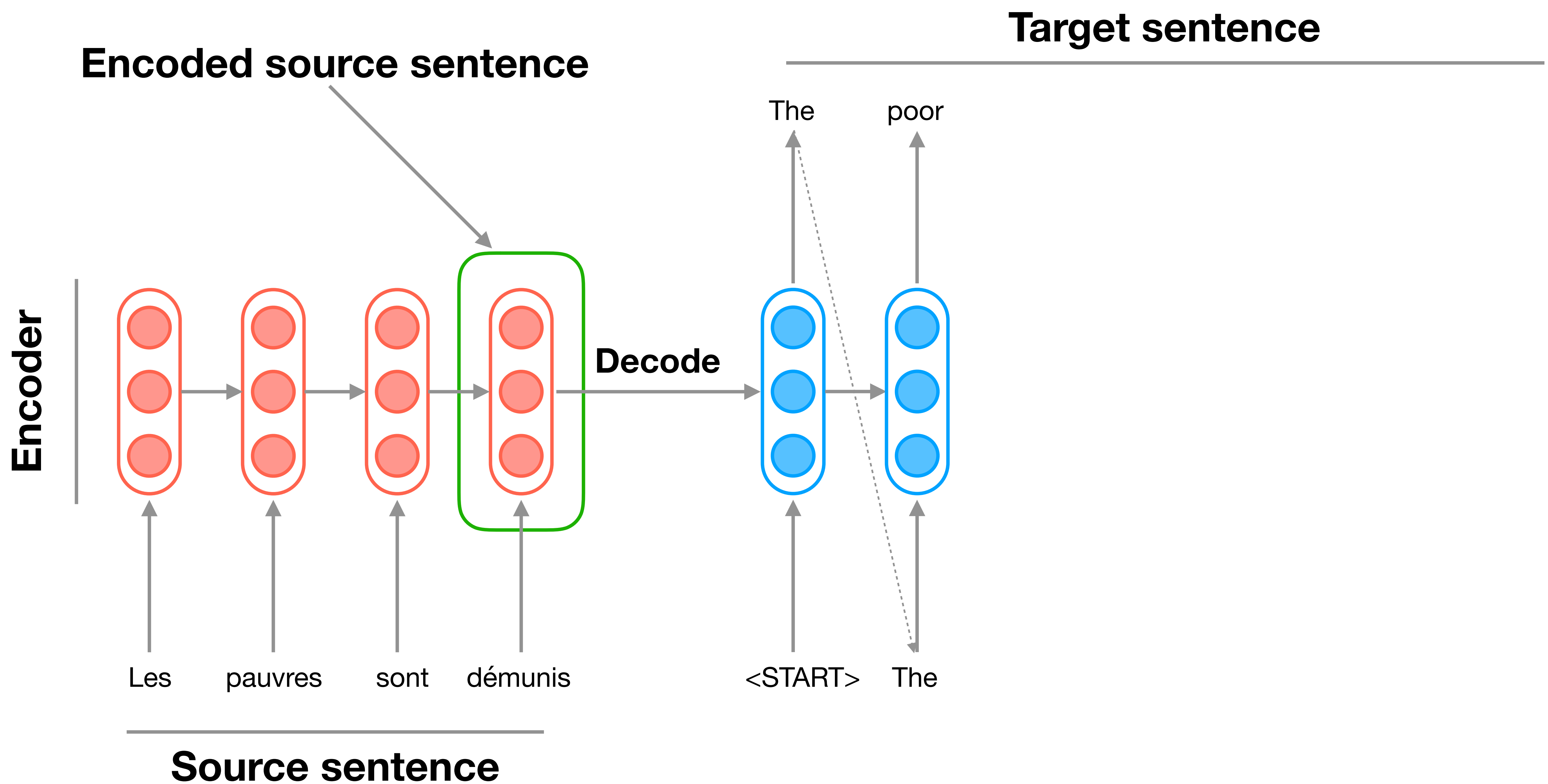
Sequence to sequence

Inference



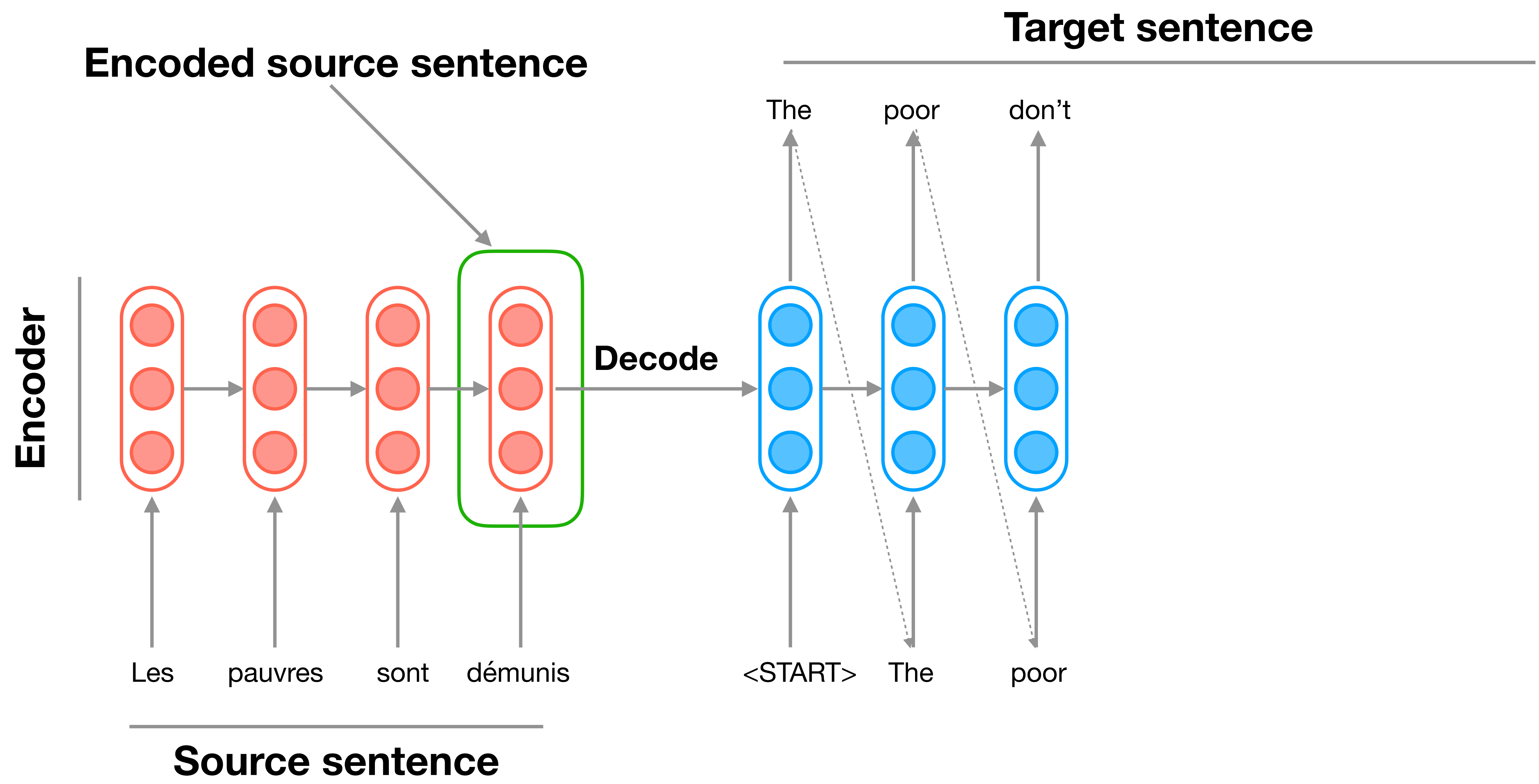
Sequence to sequence

Inference



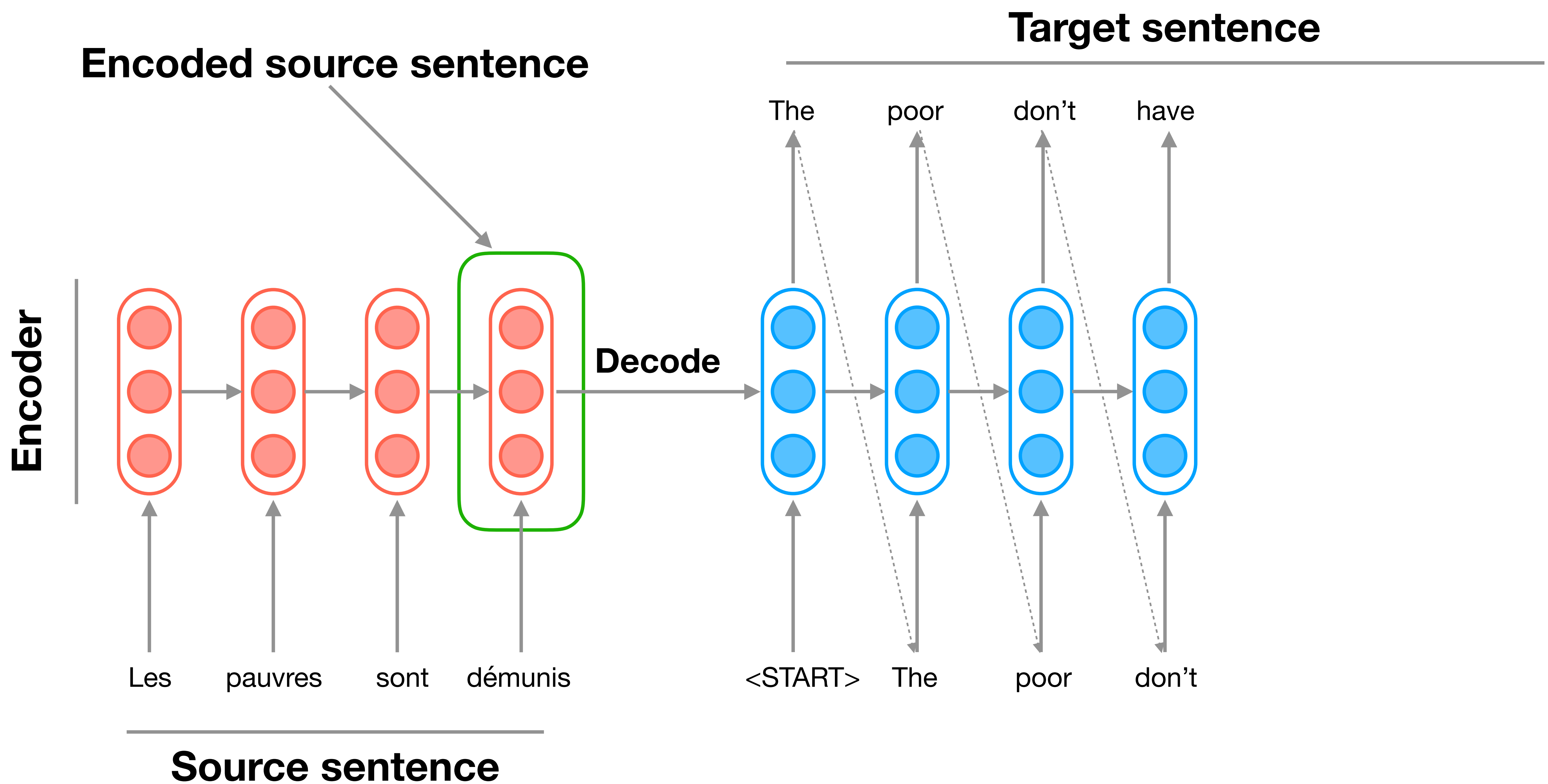
Sequence to sequence

Inference



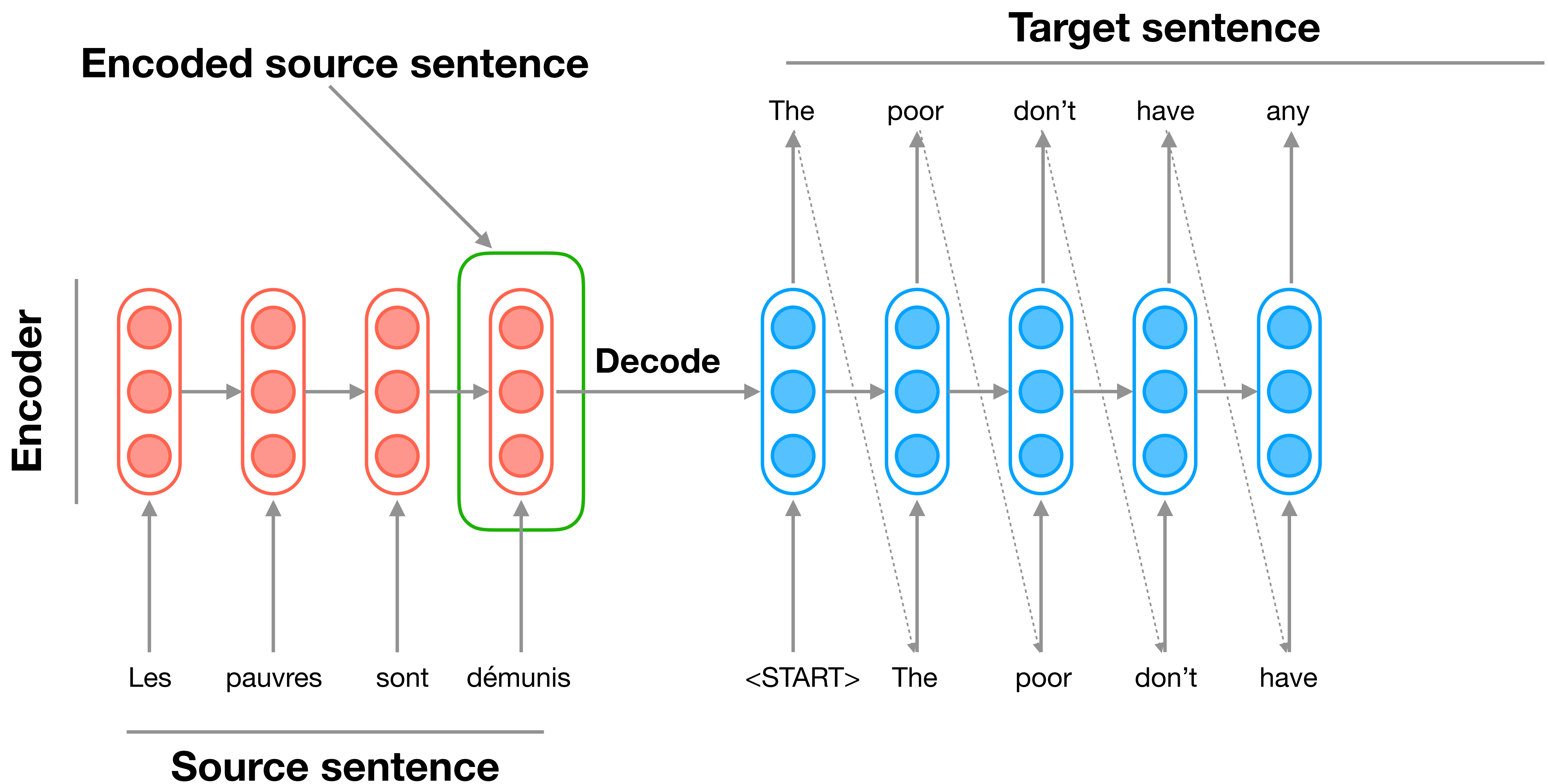
Sequence to sequence

Inference



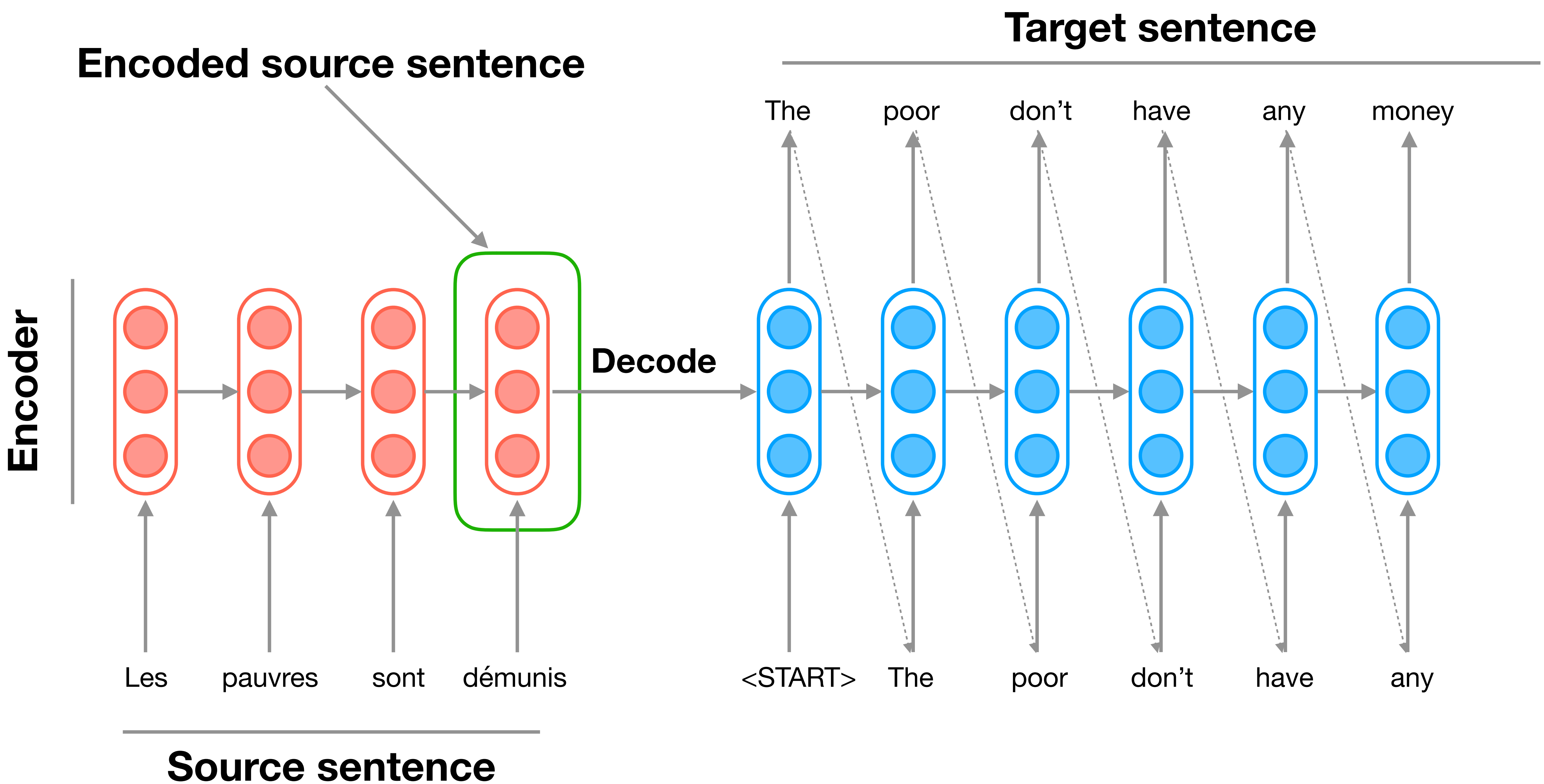
Sequence to sequence

Inference



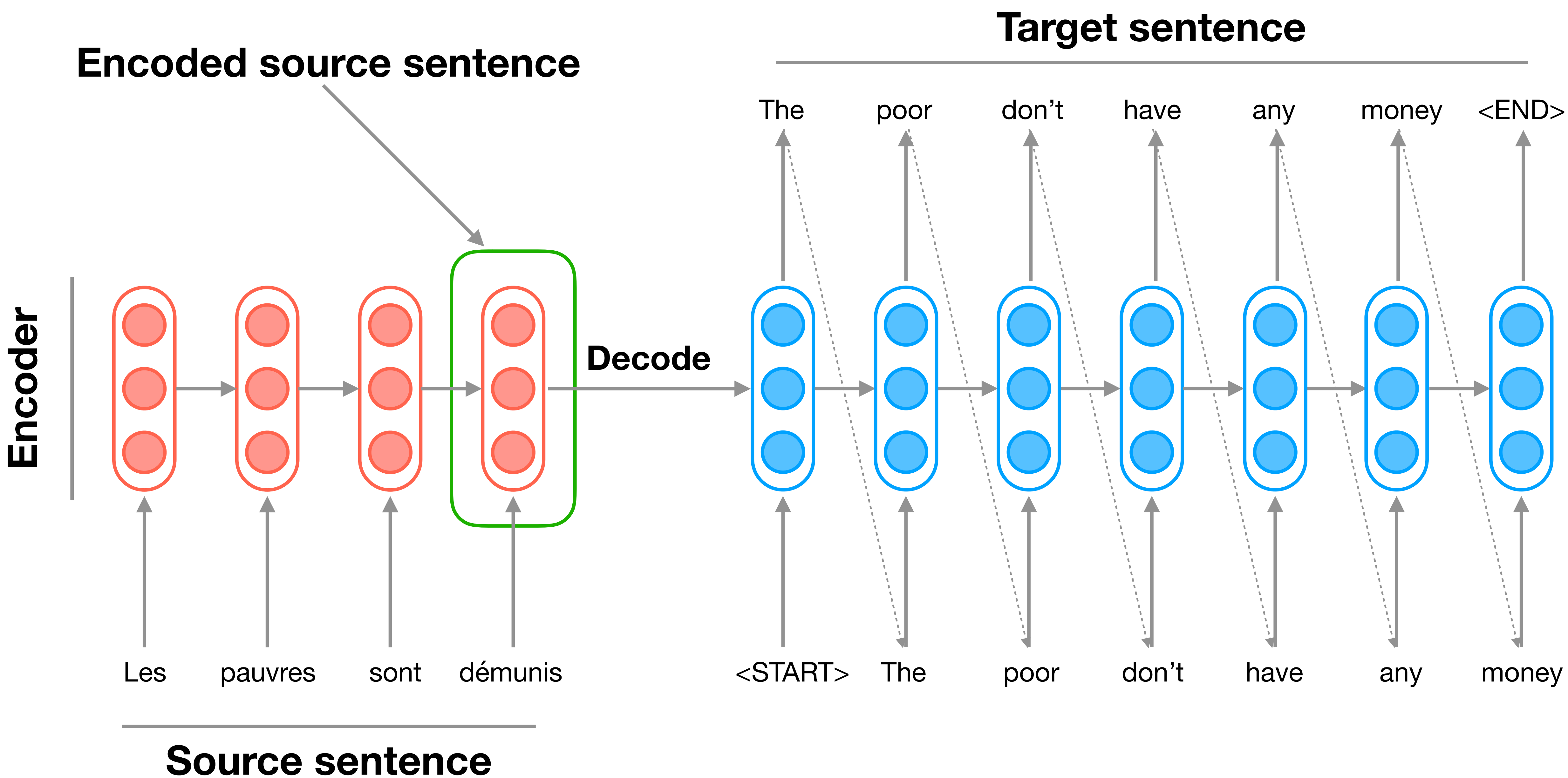
Sequence to sequence

Inference



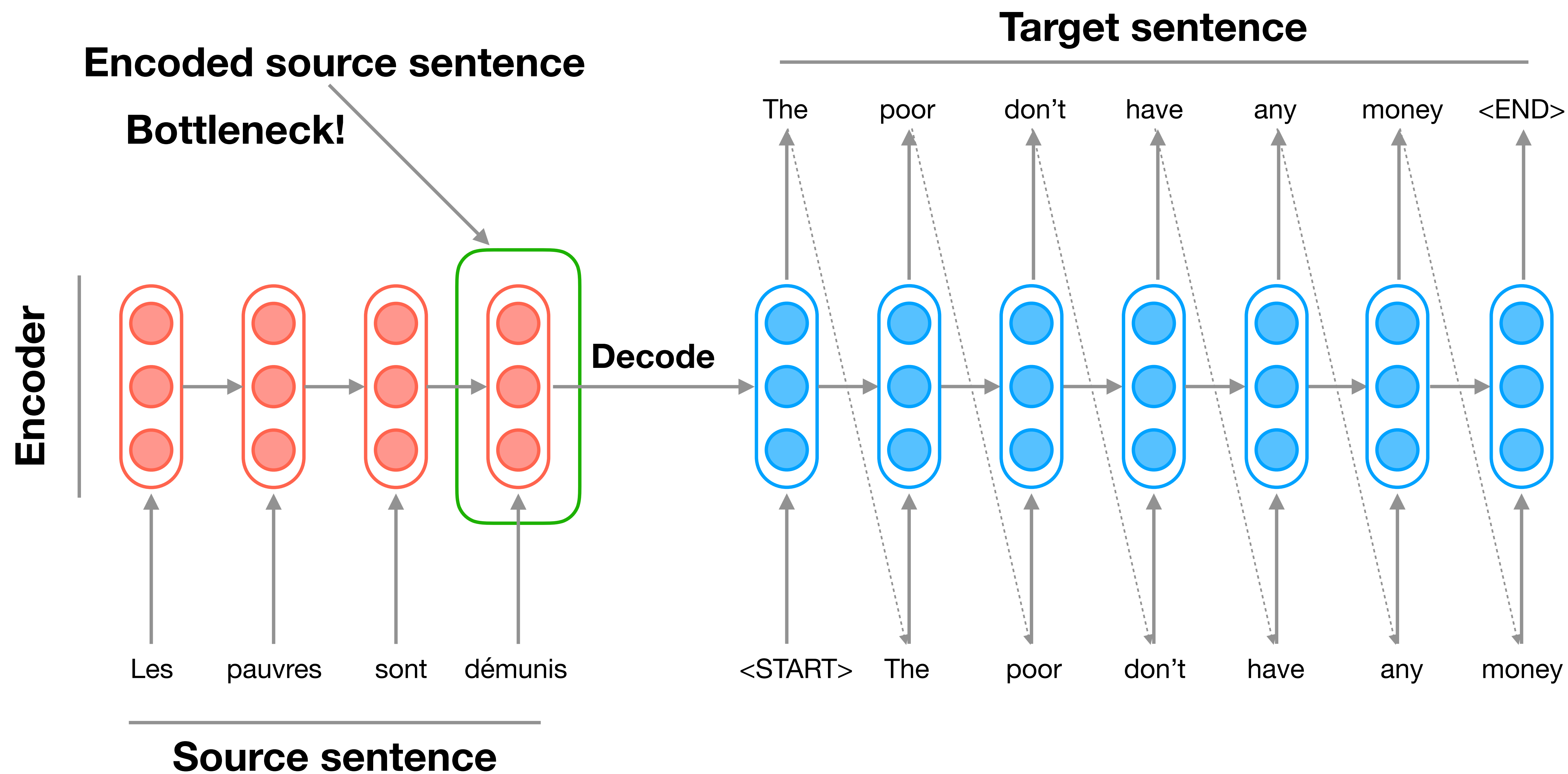
Sequence to sequence

Inference



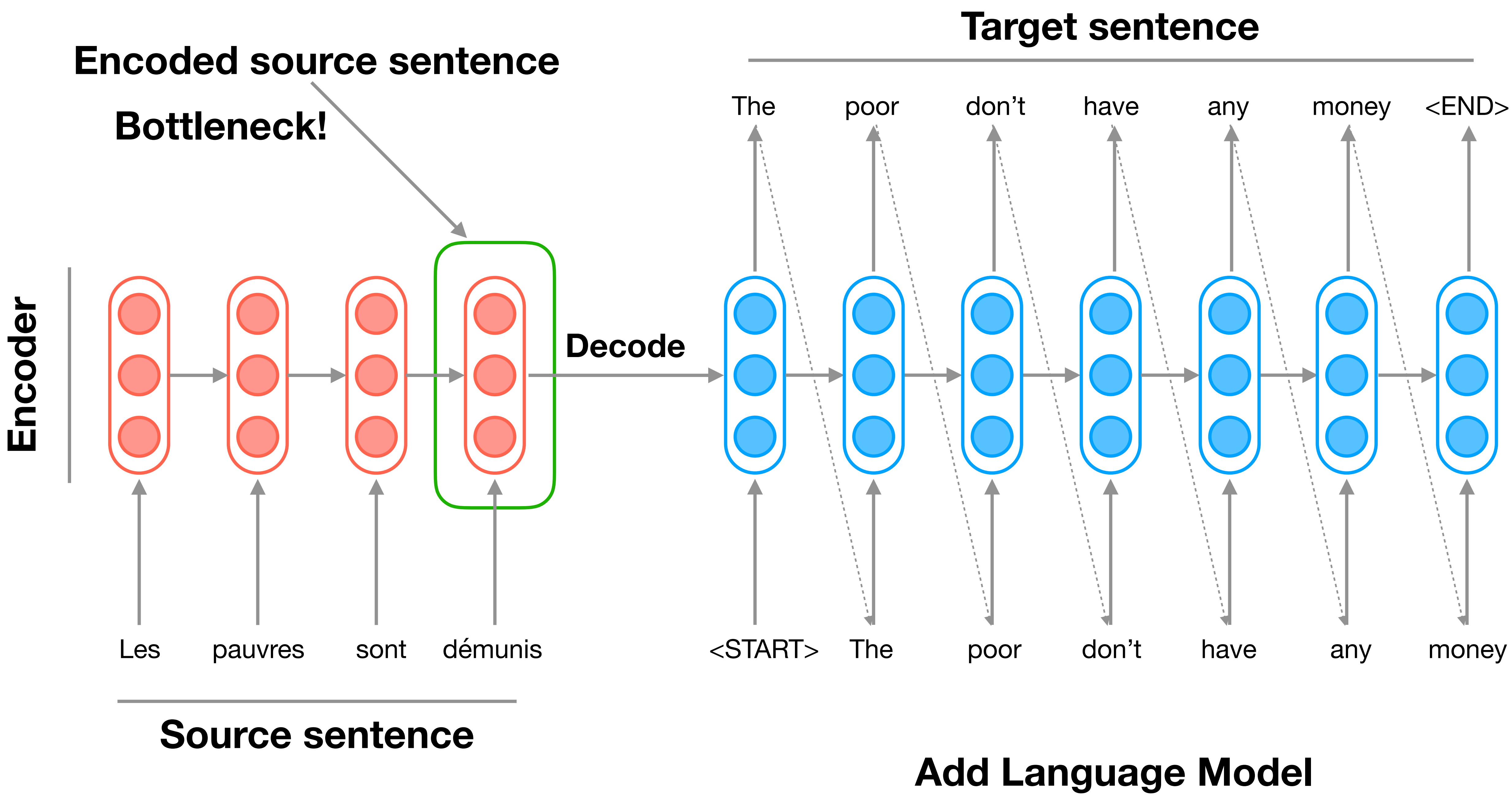
Sequence to sequence

Inference



Sequence to sequence

Inference



Contextualized Word Vectors

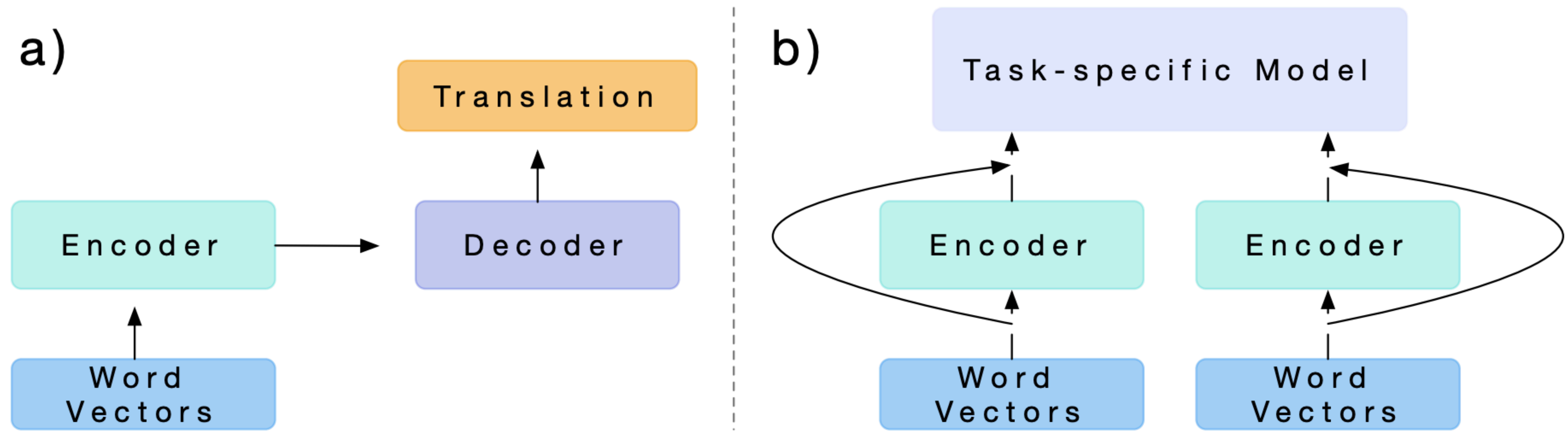


Figure 1: We a) train a two-layer, bidirectional LSTM as the encoder of an attentional sequence-to-sequence model for machine translation and b) use it to provide context for other NLP models.

Thanks for your Attention!

Boris Zubarev

@bobazooba