Neural Machine Translation

Shusen Wang

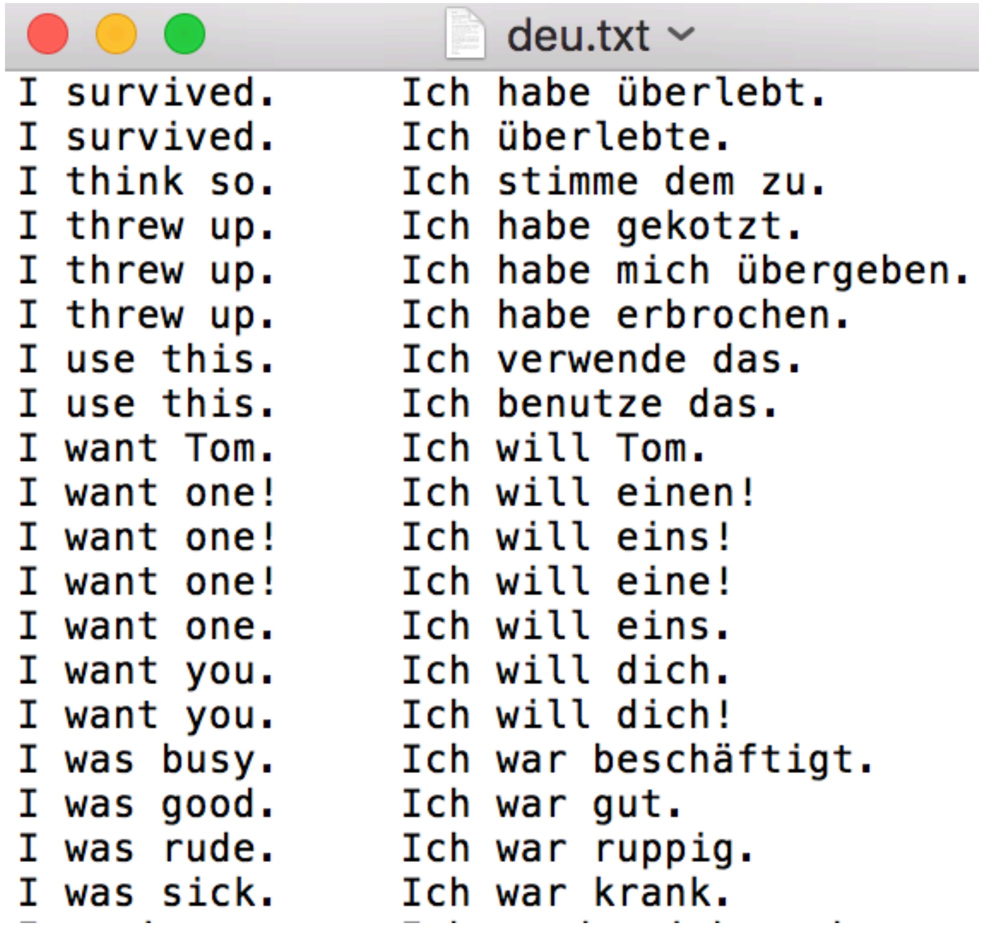
Sequence-to-Sequence Model (Seq2Seq)

English				German
“do you agree”	=>	**[Seq2Seq]**	=>	“bist du einverstanden”
“go to sleep”	=>	**[Seq2Seq]**	=>	“gehen Sie schlafen”
“We will fight”	=>	**[Seq2Seq]**	=>	“Wir werden kämpfen”

Machine Translation Data

Datasets

- Tab-delimited Bilingual Sentence Pairs: http://www.manythings.org/anki/


deu.txt

I survived.	Ich habe überlebt.
I survived.	Ich überlebte.
I think so.	Ich stimme dem zu.
I threw up.	Ich habe gekotzt.
I threw up.	Ich habe mich übergeben.
I threw up.	Ich habe erbrochen.
I use this.	Ich verwende das.
I use this.	Ich benutze das.
I want Tom.	Ich will Tom.
I want one!	Ich will einen!
I want one!	Ich will eins!
I want one!	Ich will eine!
I want one.	Ich will eins.
I want you.	Ich will dich.
I want you.	Ich will dich!
I was busy.	Ich war beschäftigt.
I was good.	Ich war gut.
I was rude.	Ich war ruppig.
I was sick.	Ich war krank.

1. Processing the Data

- **Preprocessing**: to lower case, remove punctuation, remove non-printable chars.

deu.txt

I survived.	Ich habe überlebt.
I survived.	Ich überlebte.
I think so.	Ich stimme dem zu.
I threw up.	Ich habe gekotzt.
I threw up.	Ich habe mich übergeben.
I threw up.	Ich habe erbrochen.
I use this.	Ich verwende das.
I use this.	Ich benutze das.
I want Tom.	Ich will Tom.
I want one!	Ich will einen!
I want one!	Ich will eins!
I want one!	Ich will eine!
I want one.	Ich will eins.
I want you.	Ich will dich.
I want you.	Ich will dich!
I was busy.	Ich war beschäftigt.
I was good.	Ich war gut.
I was rude.	Ich war ruppig.
I was sick.	Ich war krank.

2. Tokenization

- `input_texts  => [`**`Eng_Tokenizer`**`] => input_tokens`
- `target_texts => [`**`Deu_Tokenizer`**`] => target_tokens`

Use 2 different tokenizers for the 2 languages

2. Tokenization

- `input_texts  => [Eng_Tokenizer] => input_tokens`
- `target_texts => [Deu_Tokenizer] => target_tokens`

Tokenization in the char-level.

Tokenization in the word-level.

2. Tokenization

- `input_texts  => [Eng_Tokenizer] => input_tokens`
- `target_texts => [Deu_Tokenizer] => target_tokens`

Tokenization in the char-level.

Eng_Tokenizer

- `"I_am_okay."  => ['i', '_', 'a', 'm', ..., 'a', 'y']`

Deu_Tokenizer

- `"Es geht mir gut"  => ['e', 's', '_', ..., 'u', 't']`

2. Tokenization

Question: Why 2 different tokenizers?

Answer: In the char-level, languages have different alphabets/chars.

- English: A a, B b, C c ..., Z z. (26 letters ×2).
- German: 26 letters, 3 umlauts (Ä,Ö,Ü), and one ligature (ß).
- Greek: Α α, Β β, Γ γ, Δ δ, ..., Ω ω. (24 letters ×2).
- Chinese: 金 木 水 火 土 ... 赵 钱 孙 李 (a few thousands characters).
- Japanese: あ い う え お ... (46 Hiragana, 46 Karagana, hundreds 漢字).

2. Tokenization

Question: Why 2 different tokenizers?

Answer: In the word-level, languages have different vocabulary.

- English:

```
Machine learning is a generic term for the artificial generation of
knowledge from experience: An artificial system learns from examples
and can generalize these after completion of the learning phase.
```

- Deutsche:

```
Maschinelles Lernen ist ein Oberbegriff für die künstliche
Generierung von Wissen aus Erfahrung: Ein künstliches System lernt
aus Beispielen und kann diese nach Beendigung der Lernphase
verallgemeinern.
```

3. Encoding

Eng_Dictionary

- 'a' => 1
- 'b' => 2
- 'c' => 3
- 'd' => 4
- …
- 'z' => 26
- '_' => 27

Deu_Dictionary

- '\t' => 1 start sign
- '\n' => 2 stop sign
- 'a' => 3
- 'b' => 4
- 'c' => 5
- 'd' => 6
- …
- 'z' => 28
- '_' => 29

3. Encoding

```
"I_am_okay."
```

Eng_Tokenizer

```
['i', '_', 'a', 'm', '_', 'o', 'k', 'a', 'y']
```

Encoding using Eng_Dictionary

```
[ 9,   27,  1,   13,  27,  15,  11,  1,   25]
```

3. Encoding

"Es geht mir gut”

Deu_Tokenizer

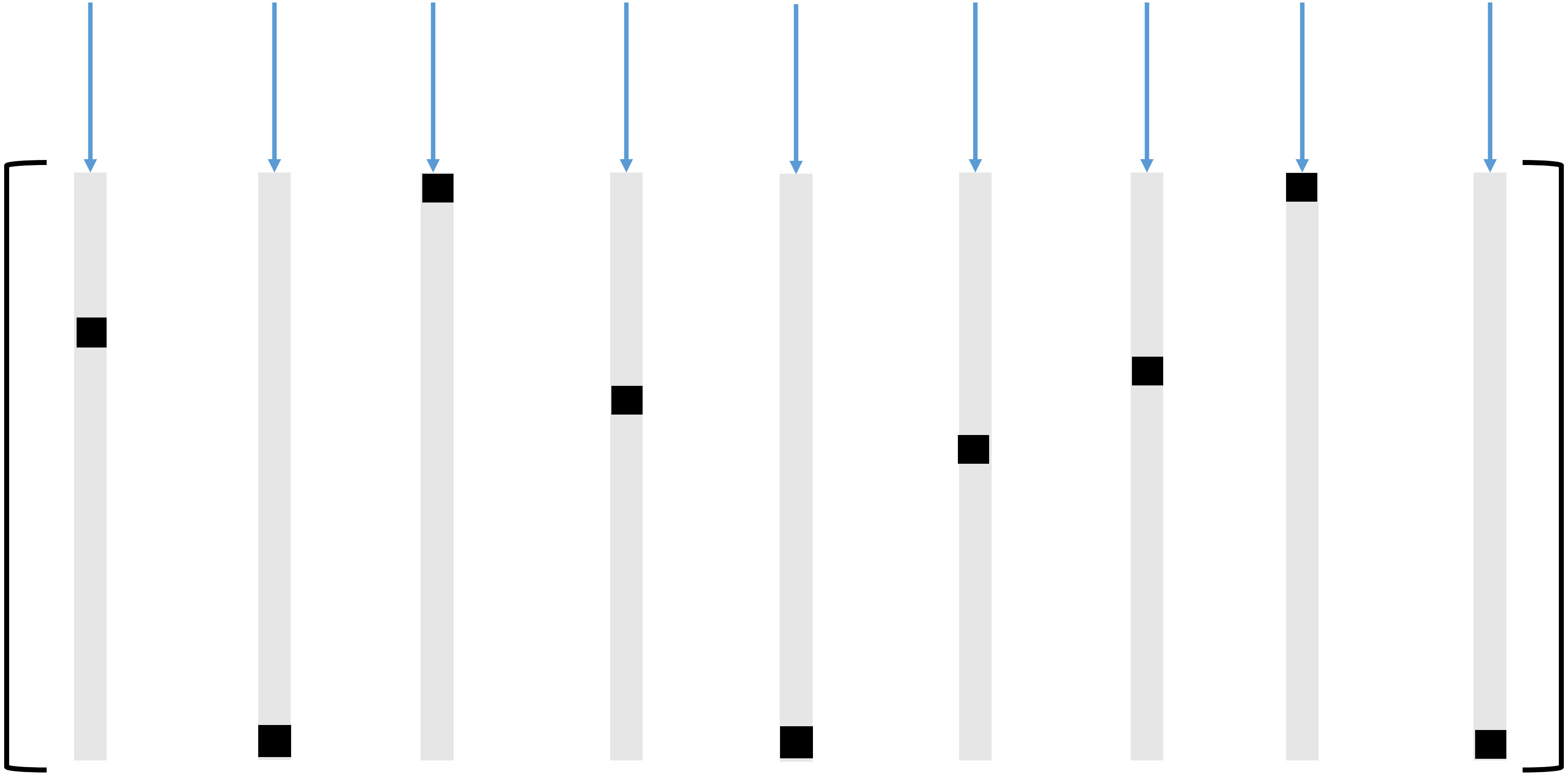
['e', 's', '_', 'g', 'e', ..., 'g', 'u', 't']

Encoding using Deu_Dictionary

[7, 21, 29, 9, 7, ..., 9, 23, 22]

4. One-Hot Encoding (Char to Vector)

```
['i', '_', 'a', 'm', '_', 'o', 'k', 'a', 'y']
[ 9,   27,  1,   13,  27,  15,  11,  1,   25]
```



4. One-Hot Encoding v.s. Embedding

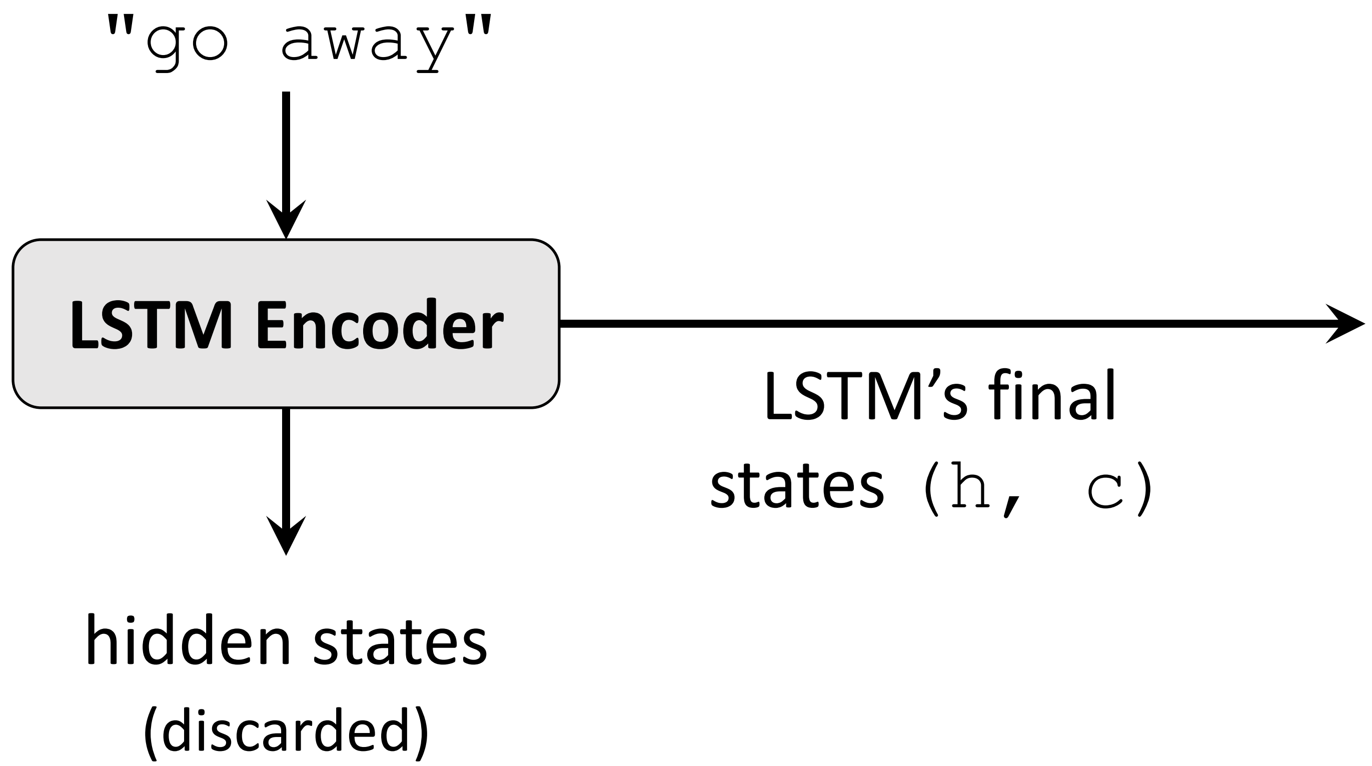
- There are around 100 frequent characters. (Vocabulary ≈ 100.)
- There are around $10K$ frequent English words. (Vocabulary $\approx 10K$.)

- One-hot converts a character to 100-dim vector.
- One-hot converts a word to $10K$-dim vector. (Too big.)

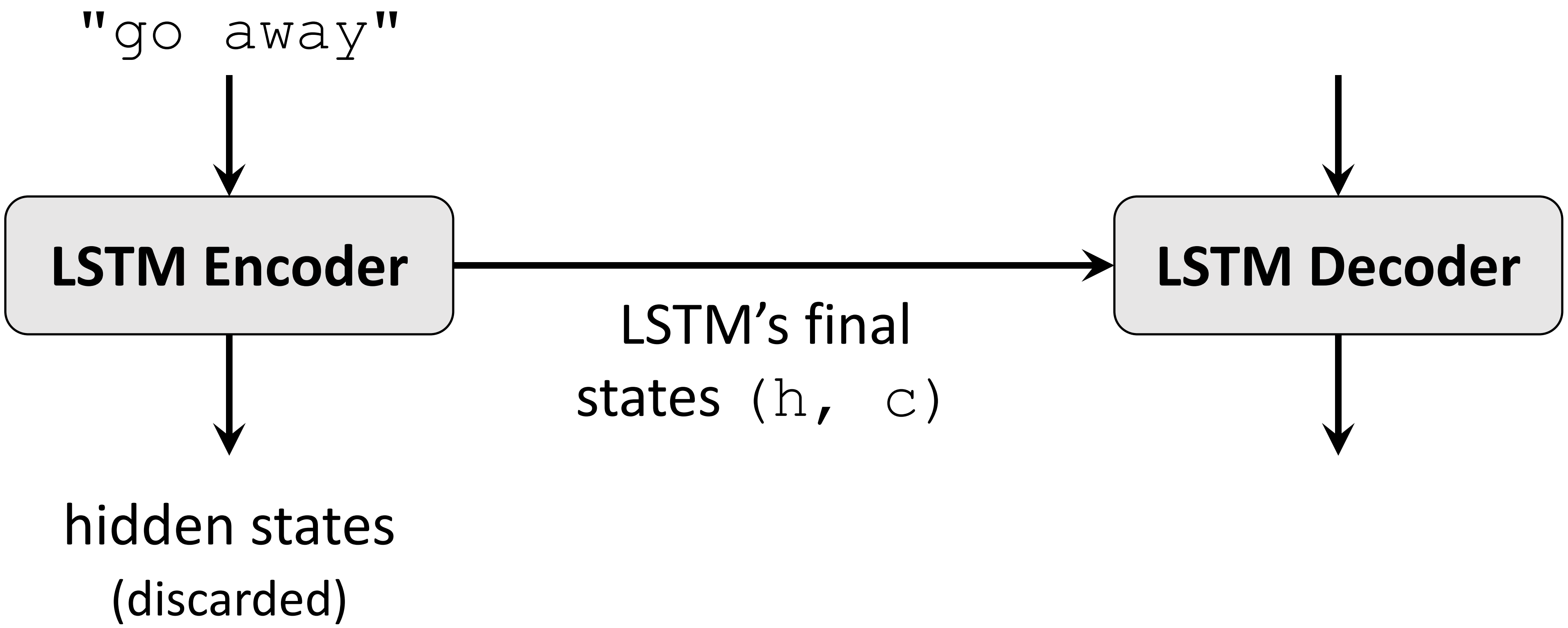
- Conclusion:
  - For char-level tokenization, use one-hot.
  - For word-level tokenization, use an embedding layer.

Seq2Seq Model: Training

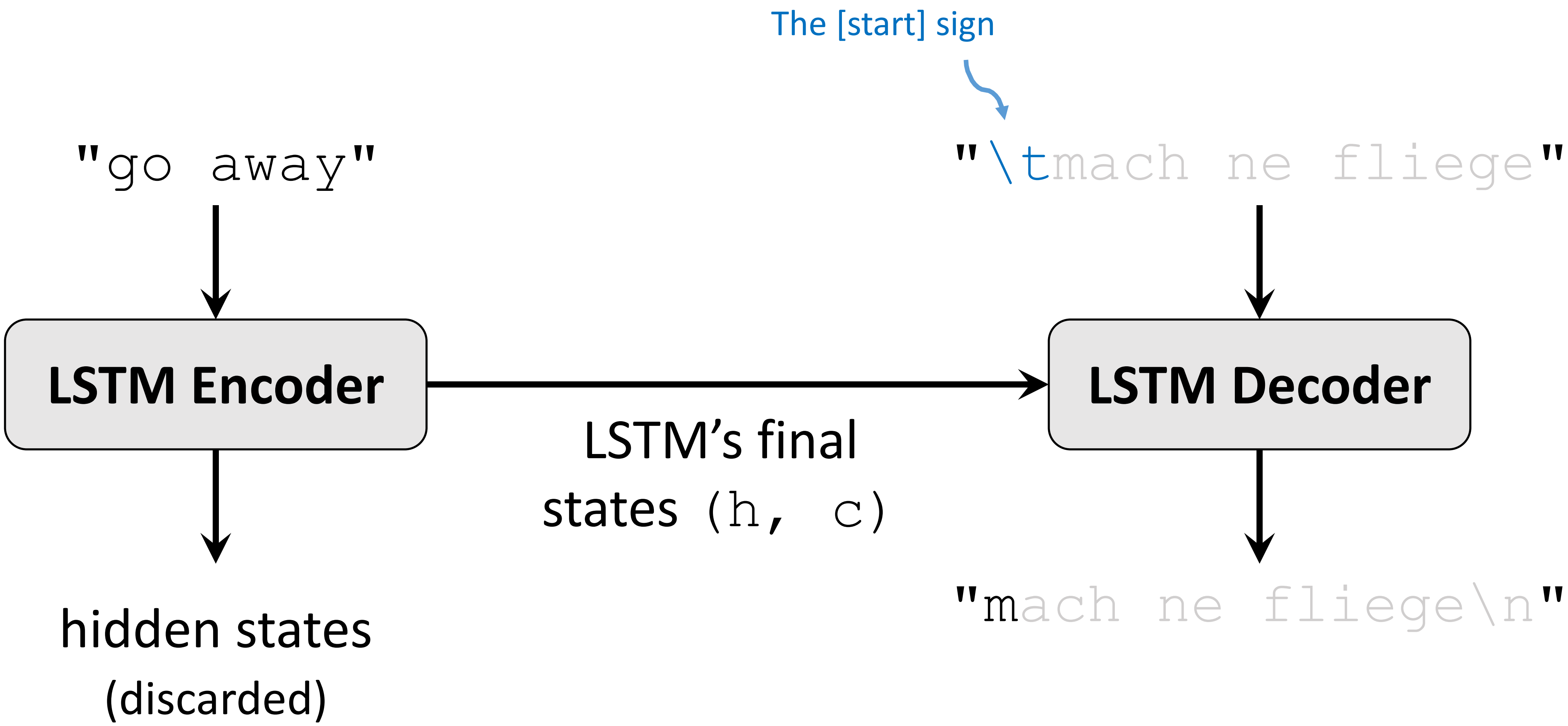
Training of the Seq2Seq Model



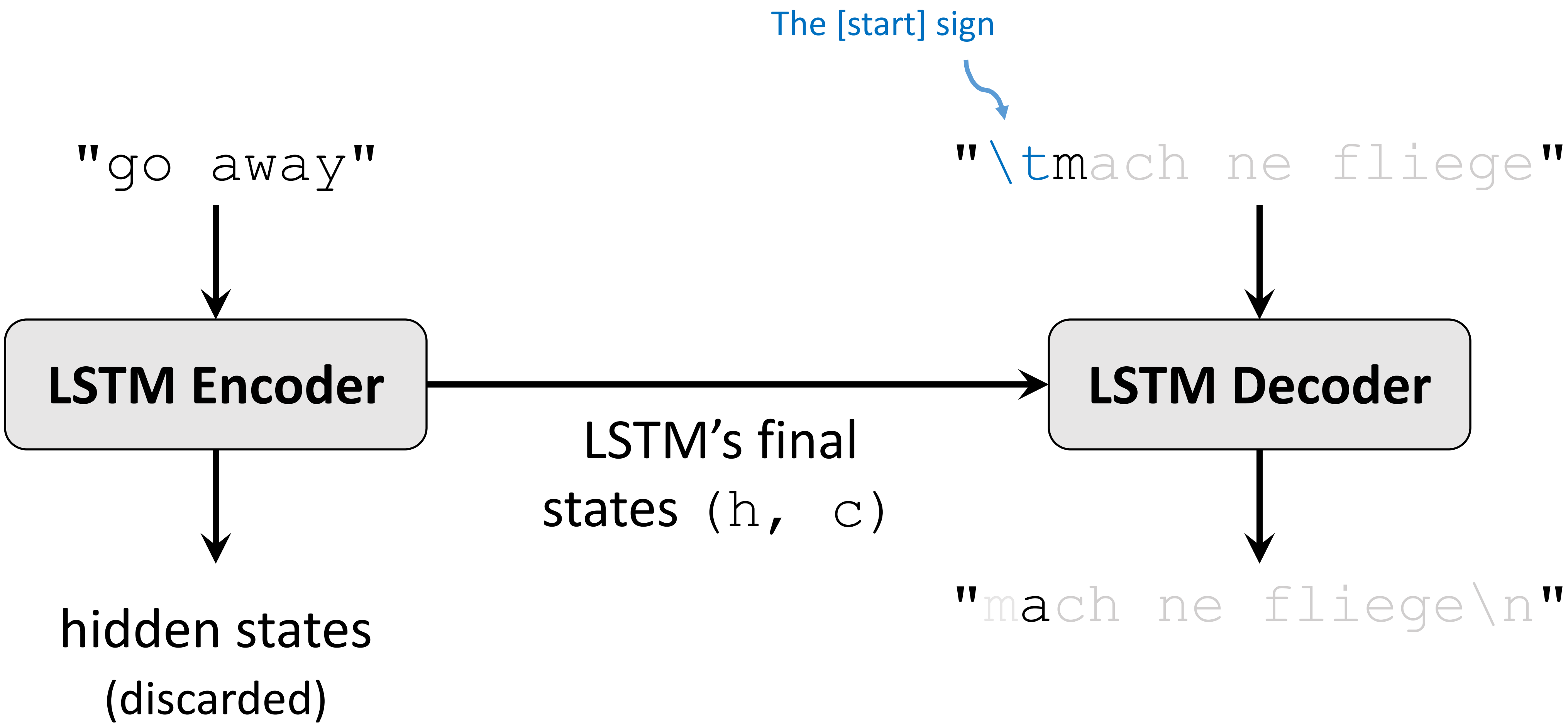
Training of the Seq2Seq Model



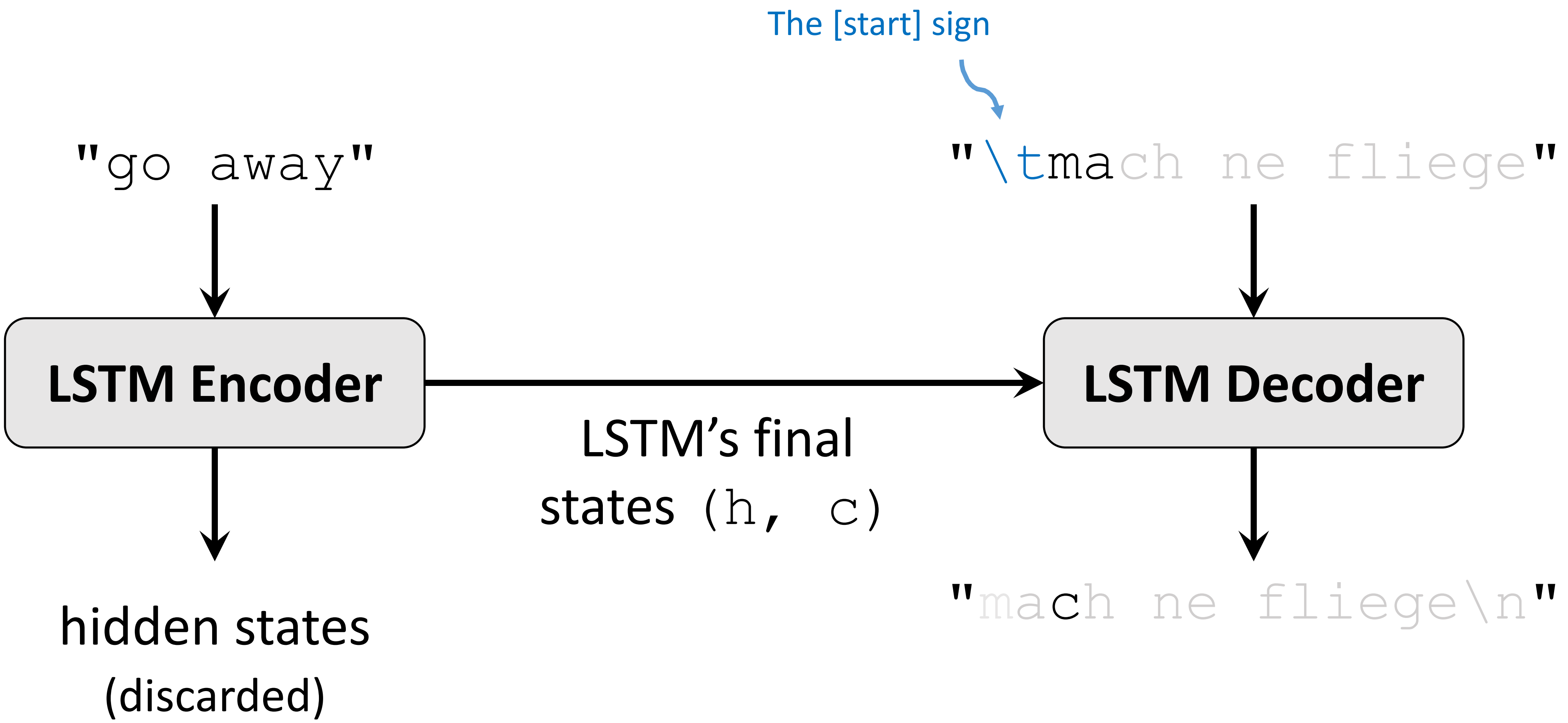
Training of the Seq2Seq Model



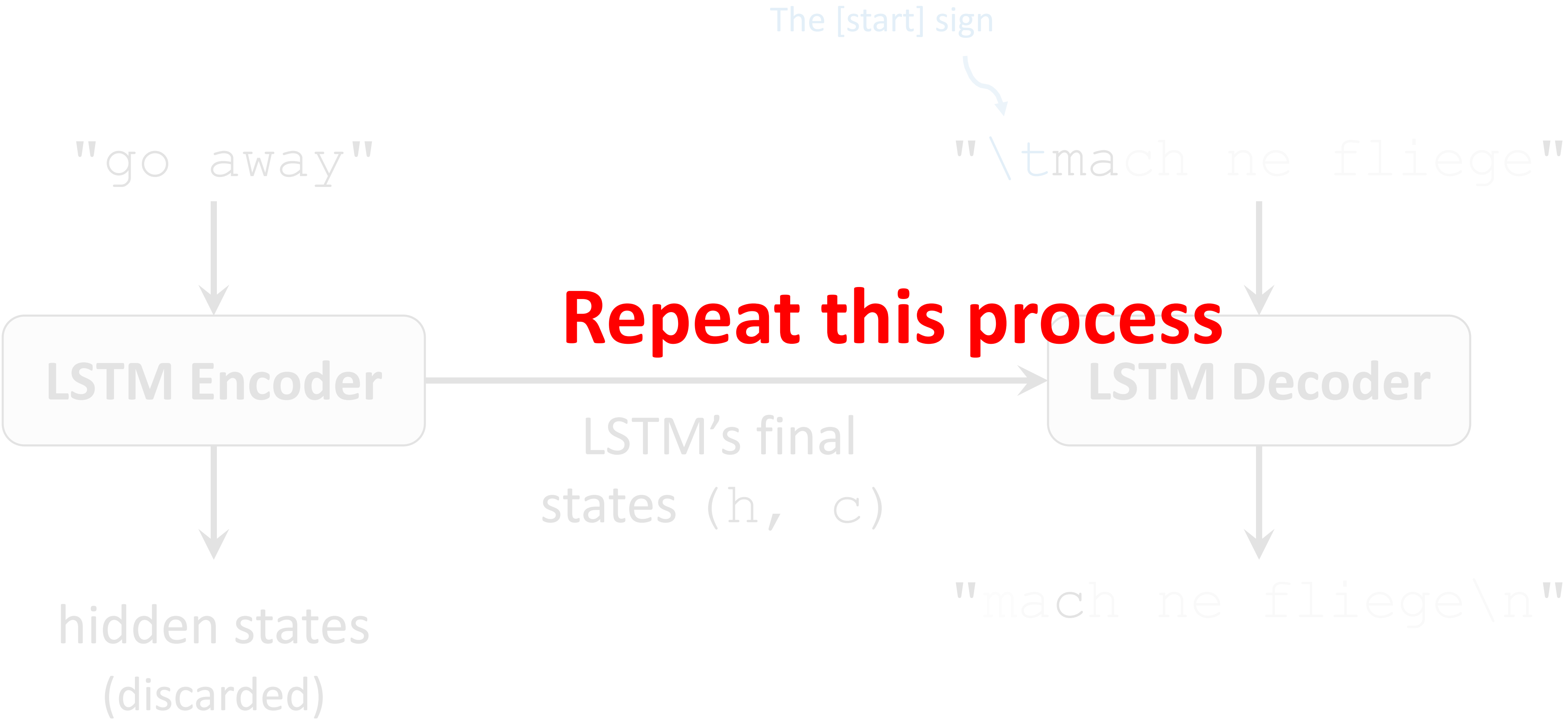
Training of the Seq2Seq Model



Training of the Seq2Seq Model

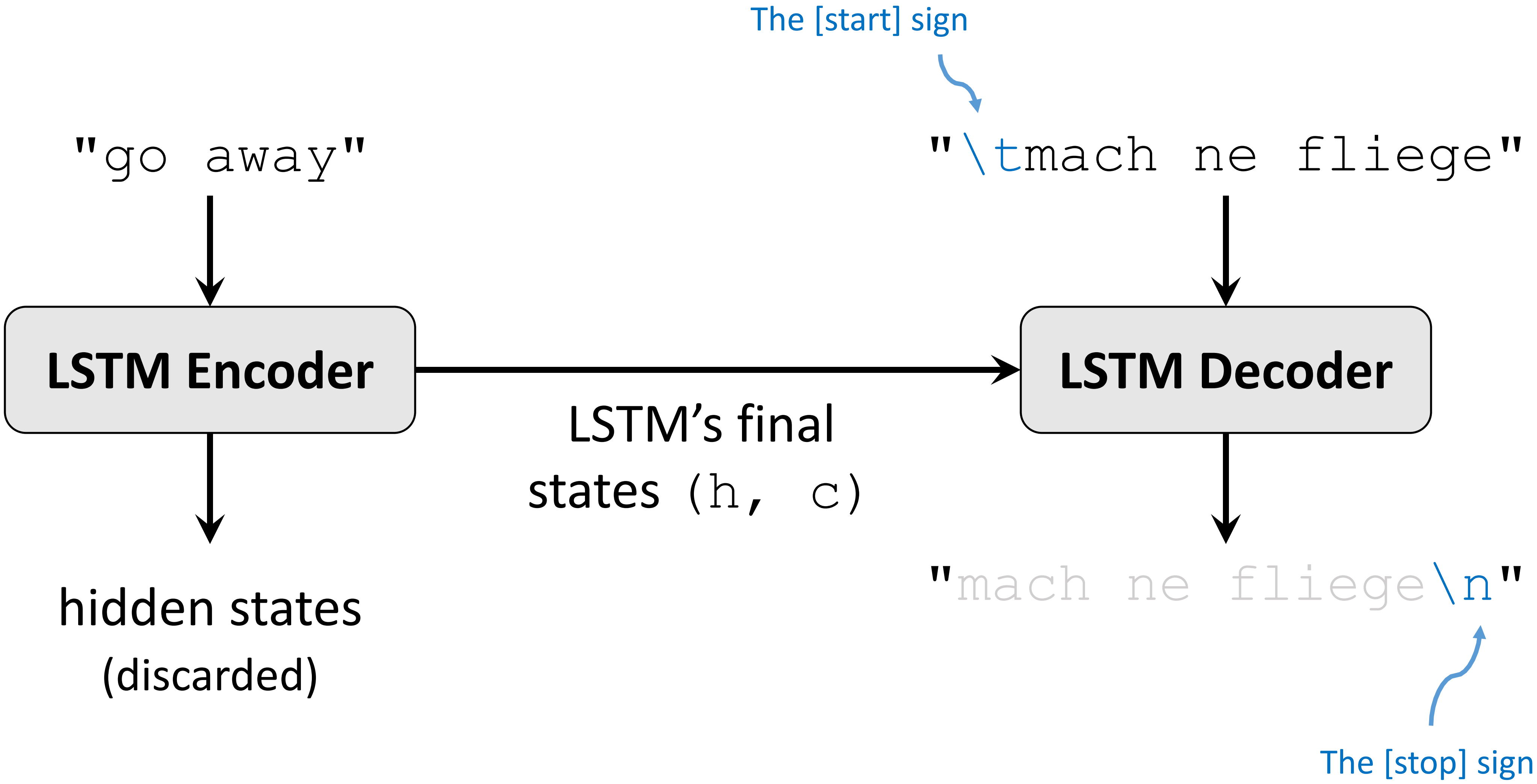


Training of the Seq2Seq Model

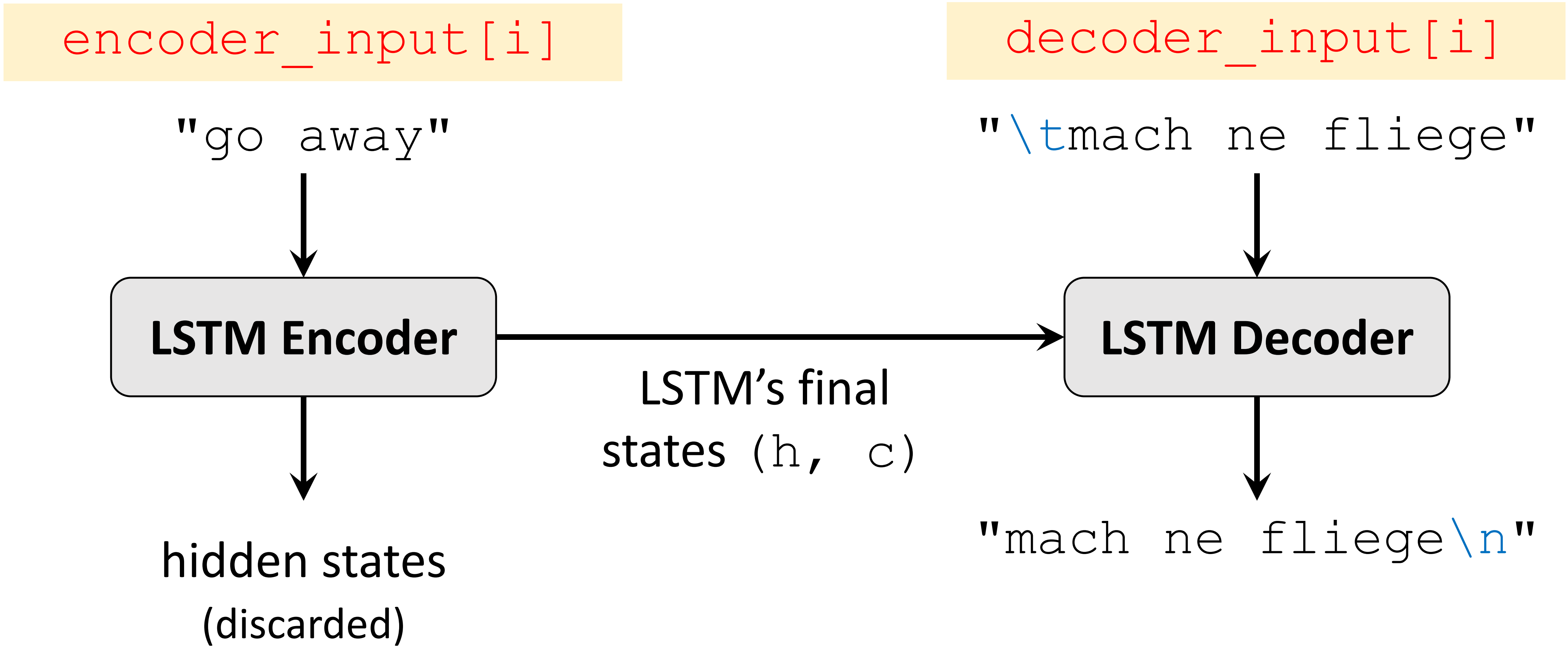


Repeat this process

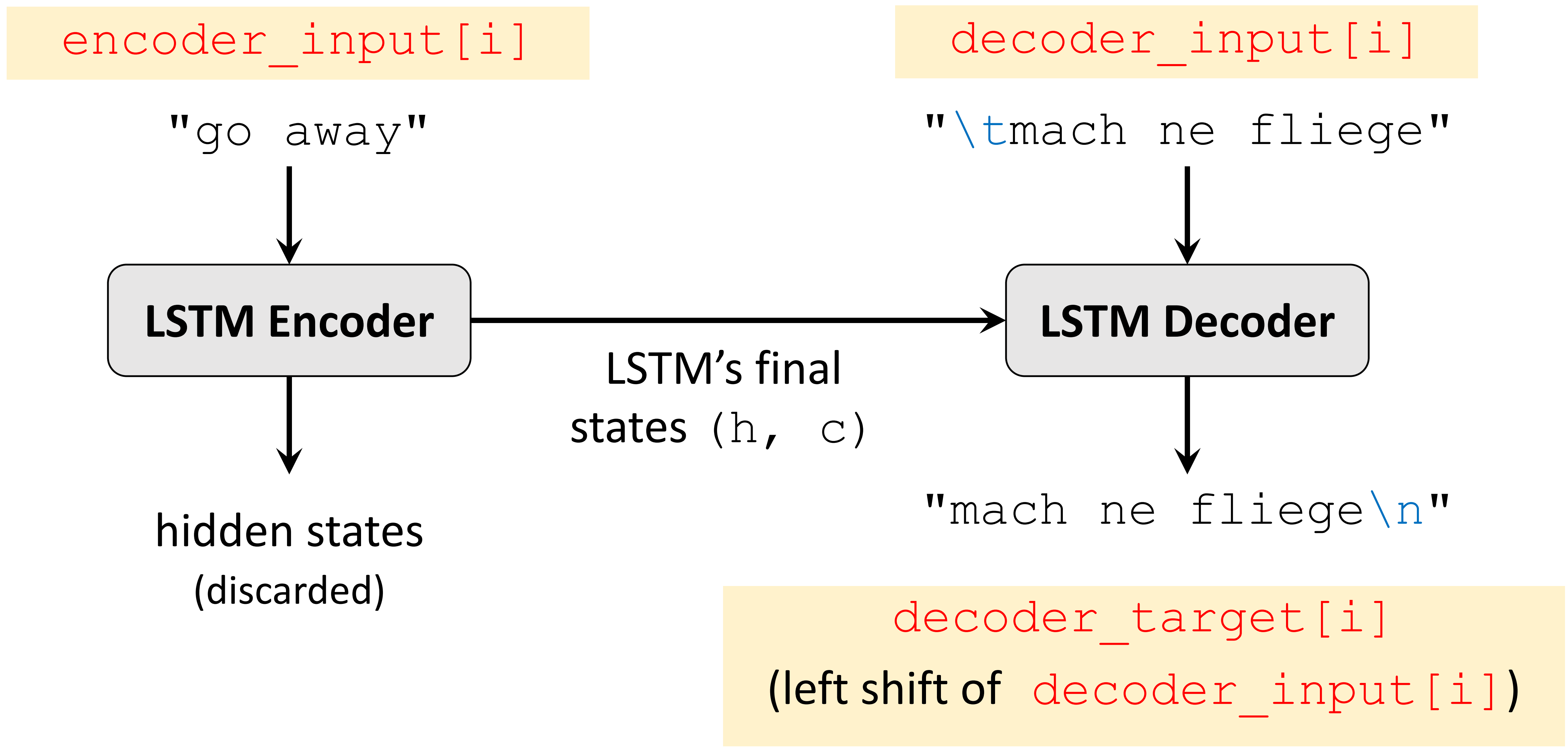
Training of the Seq2Seq Model



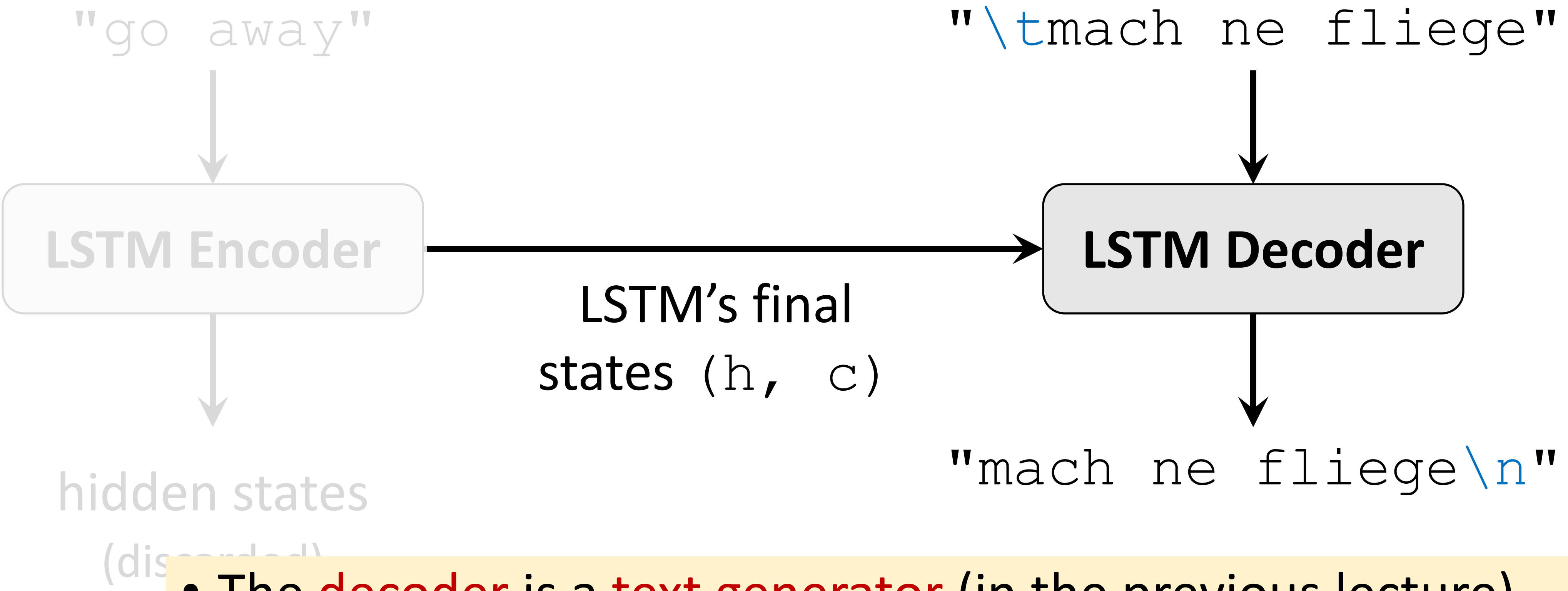
Training of the Seq2Seq Model



Training of the Seq2Seq Model

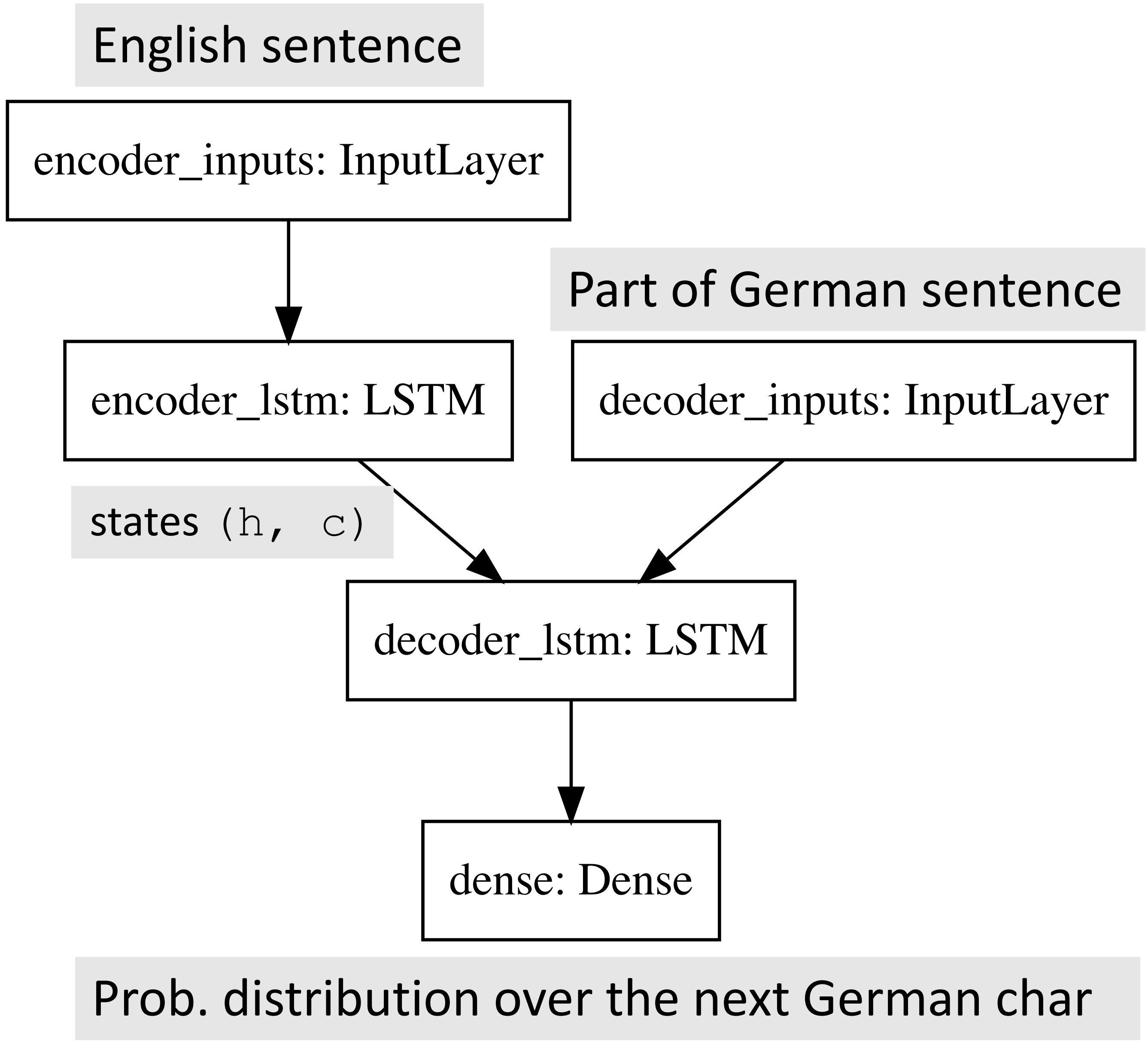


Training of the Seq2Seq Model



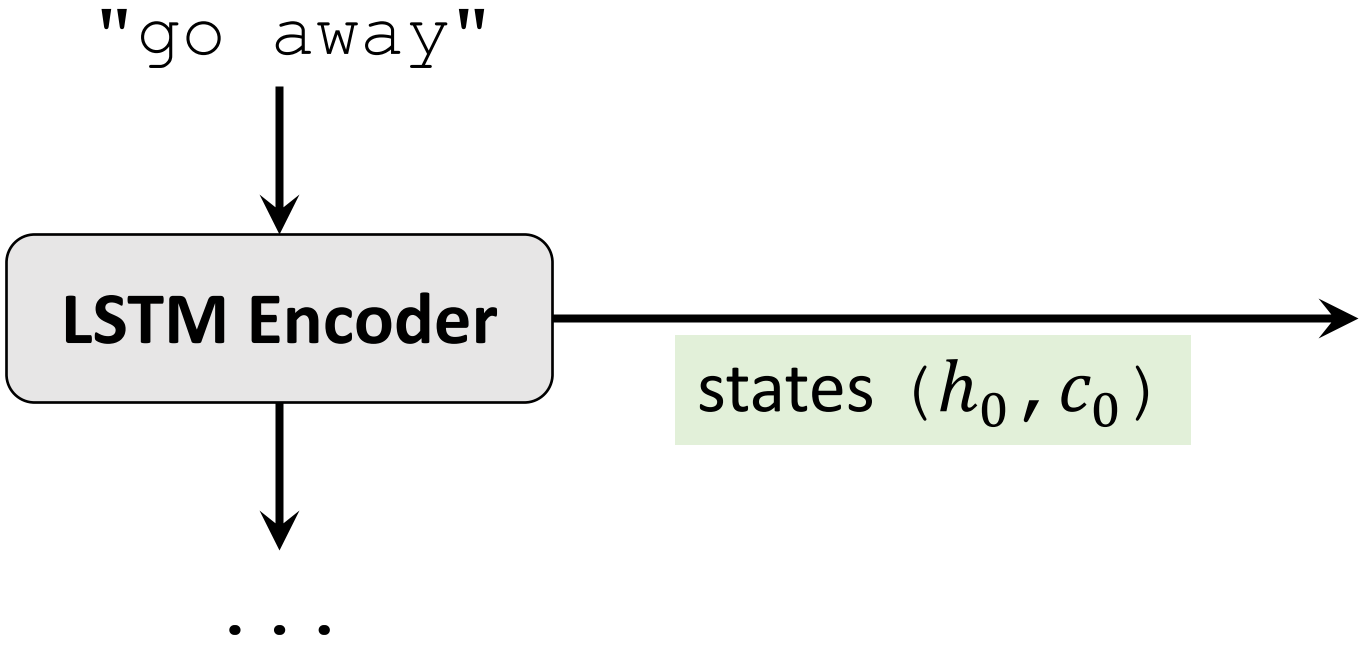
- The decoder is a text generator (in the previous lecture).
- Difference from the simple text generator: the initial states are determined by the encoder.

Seq2Seq Model in Keras

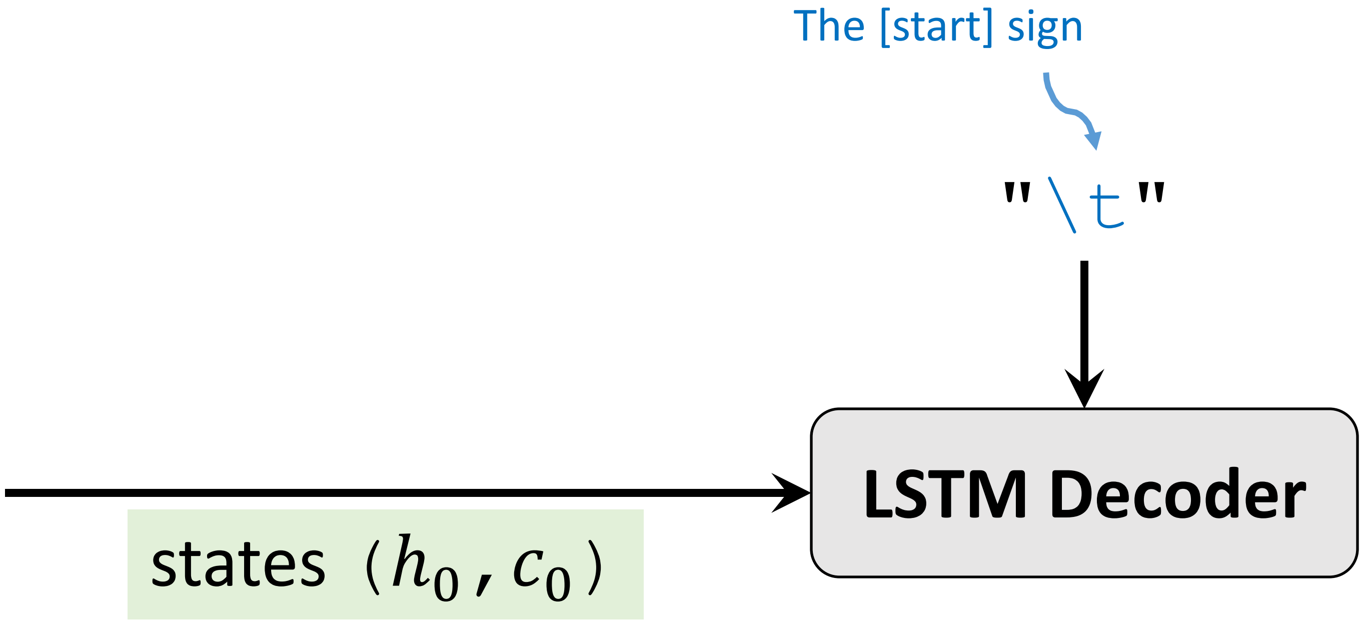


Inference Using the Seq2Seq Model

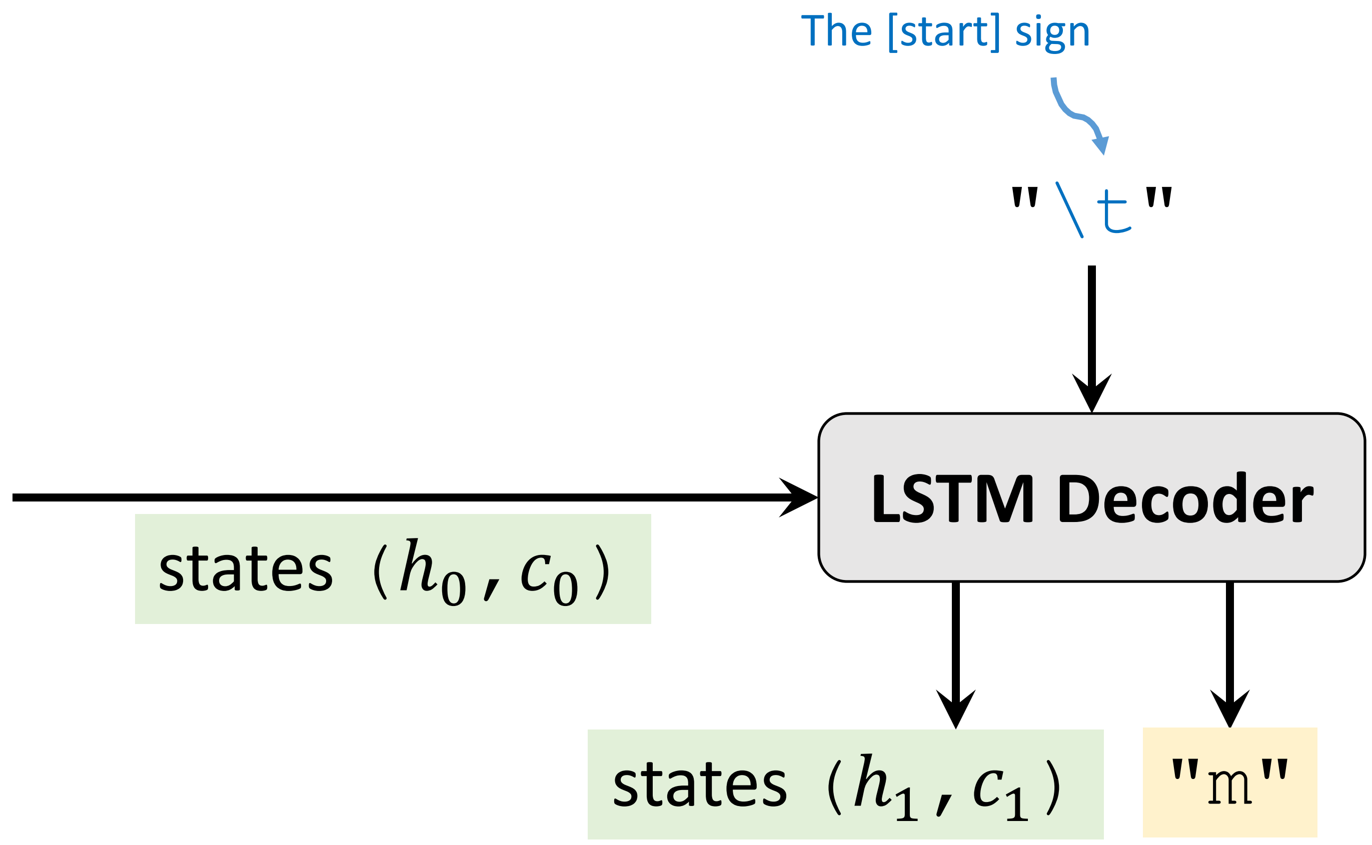
Inference



Inference

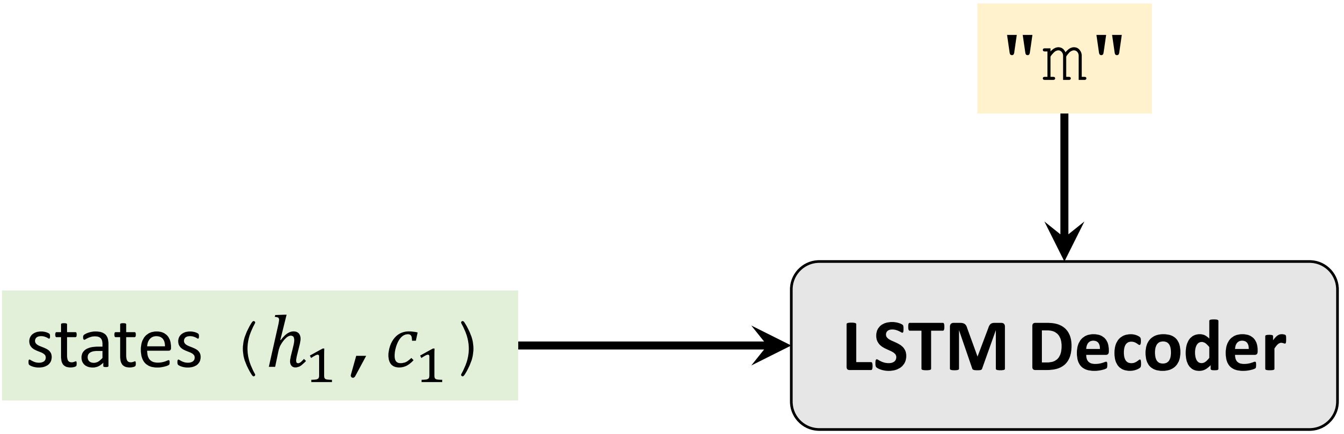


Inference



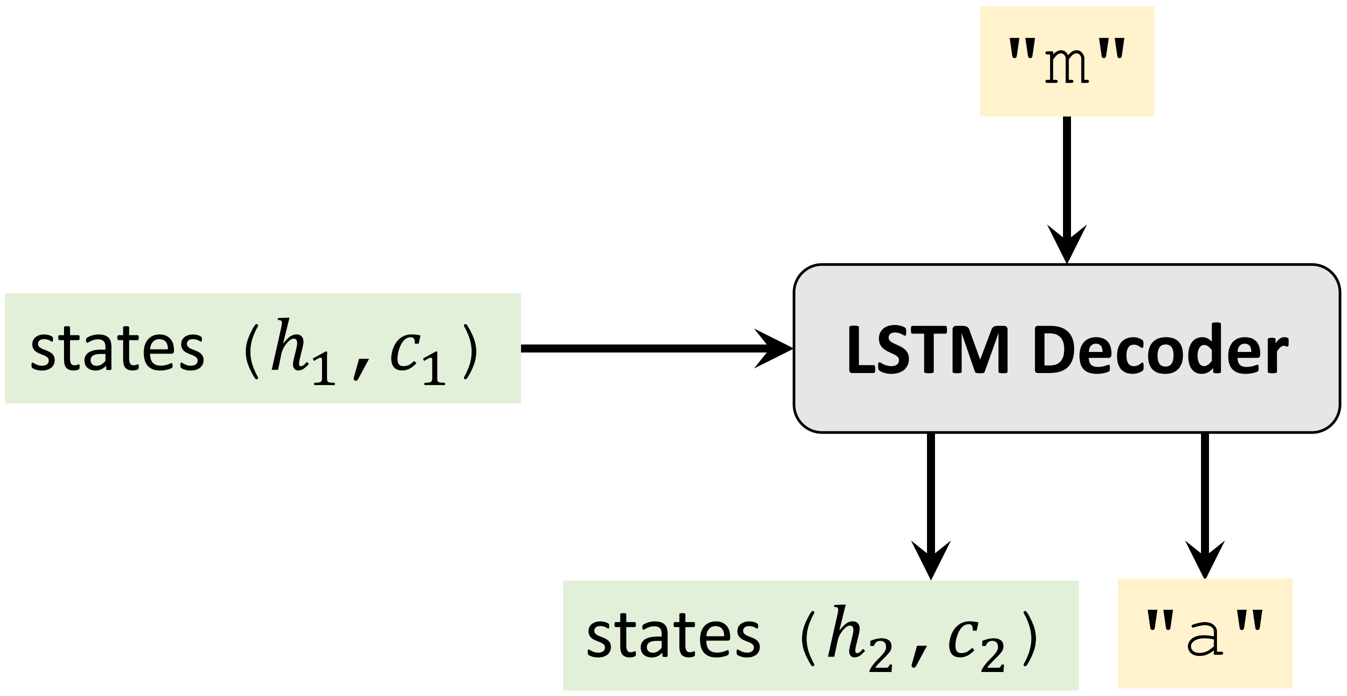
Record: "m"

Inference



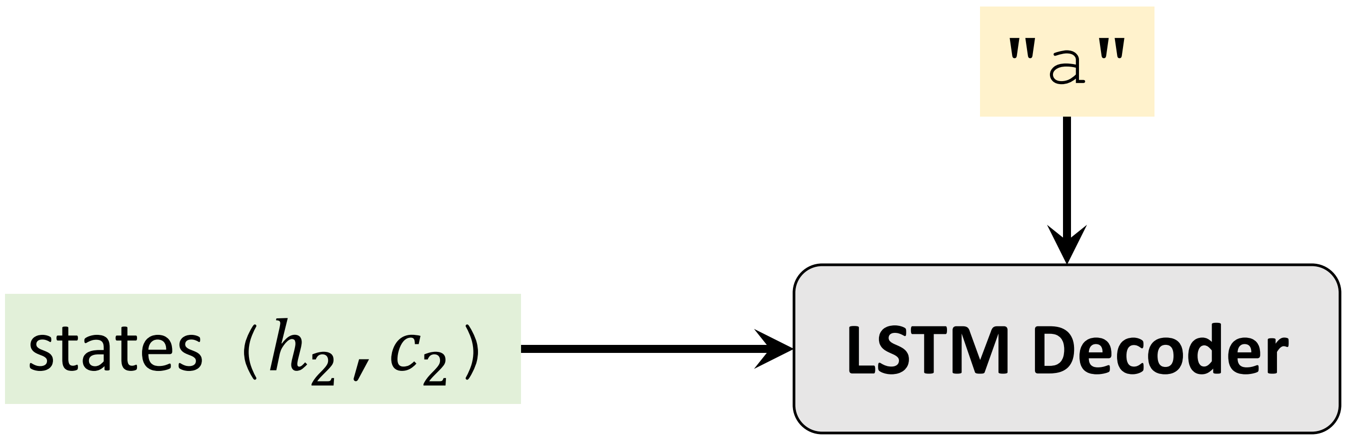
Record: "m"

Inference



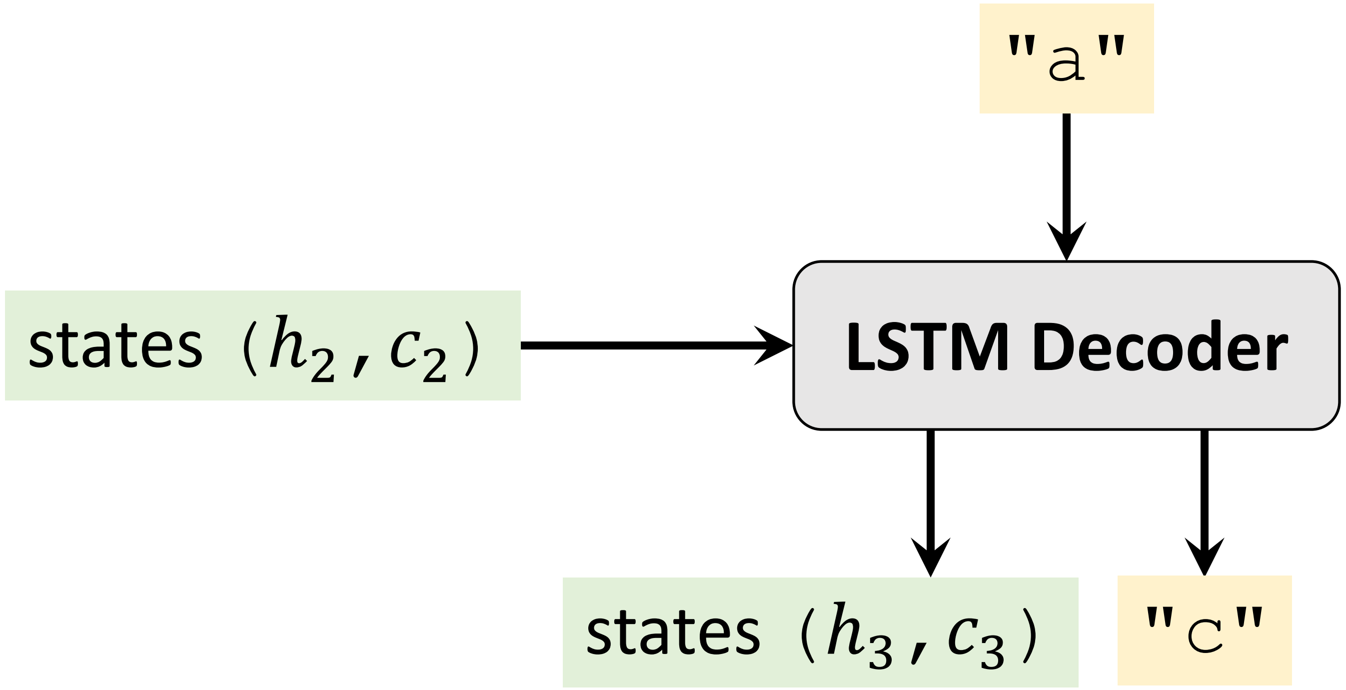
Record: `"ma"`

Inference



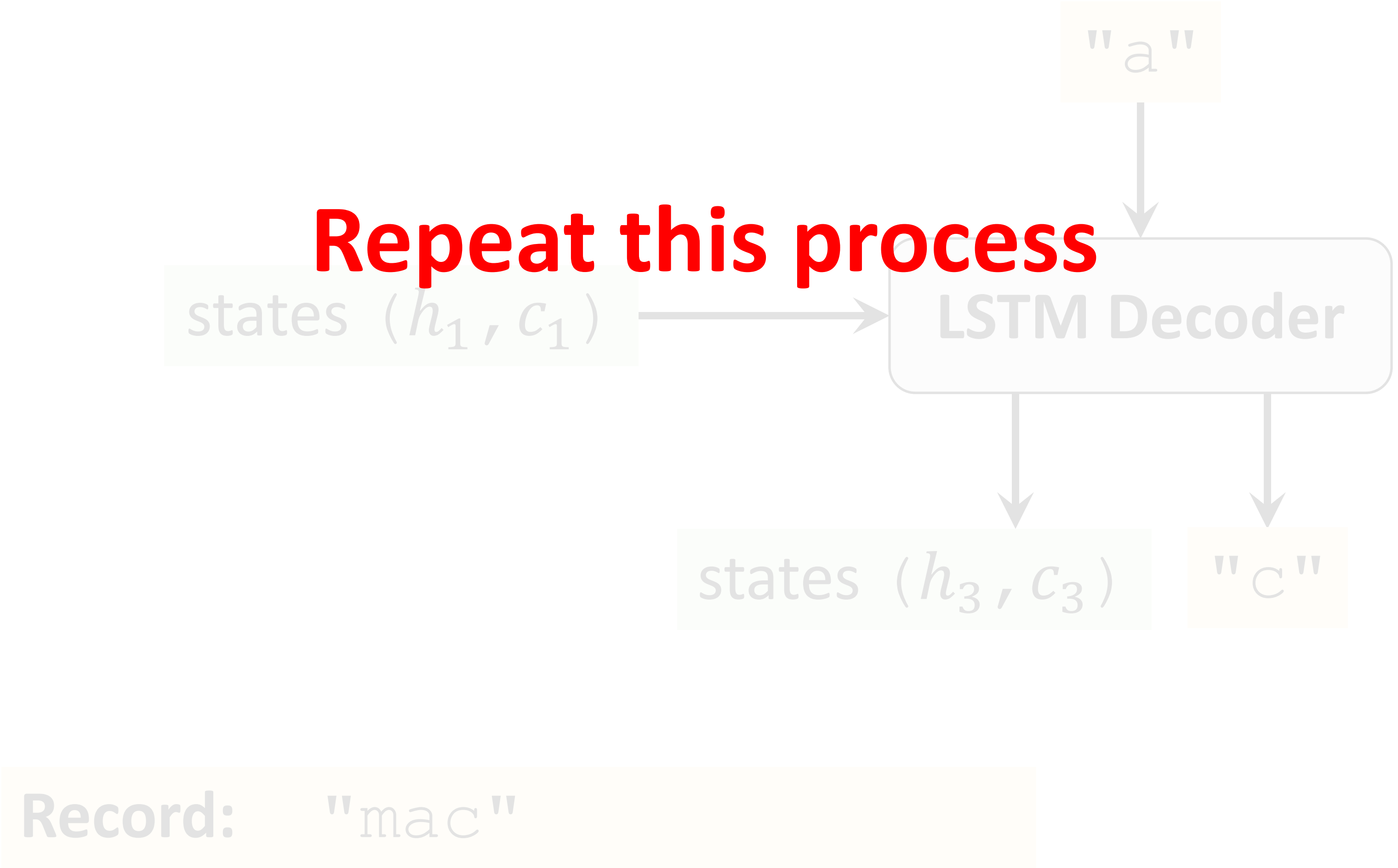
Record: "ma"

Inference

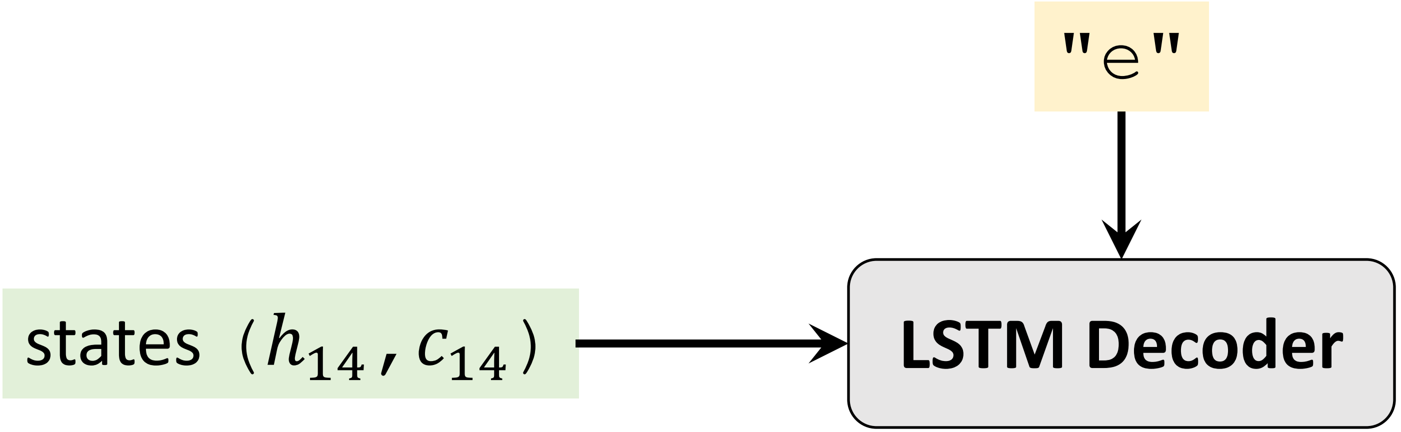


Record: `"mac"`

Inference

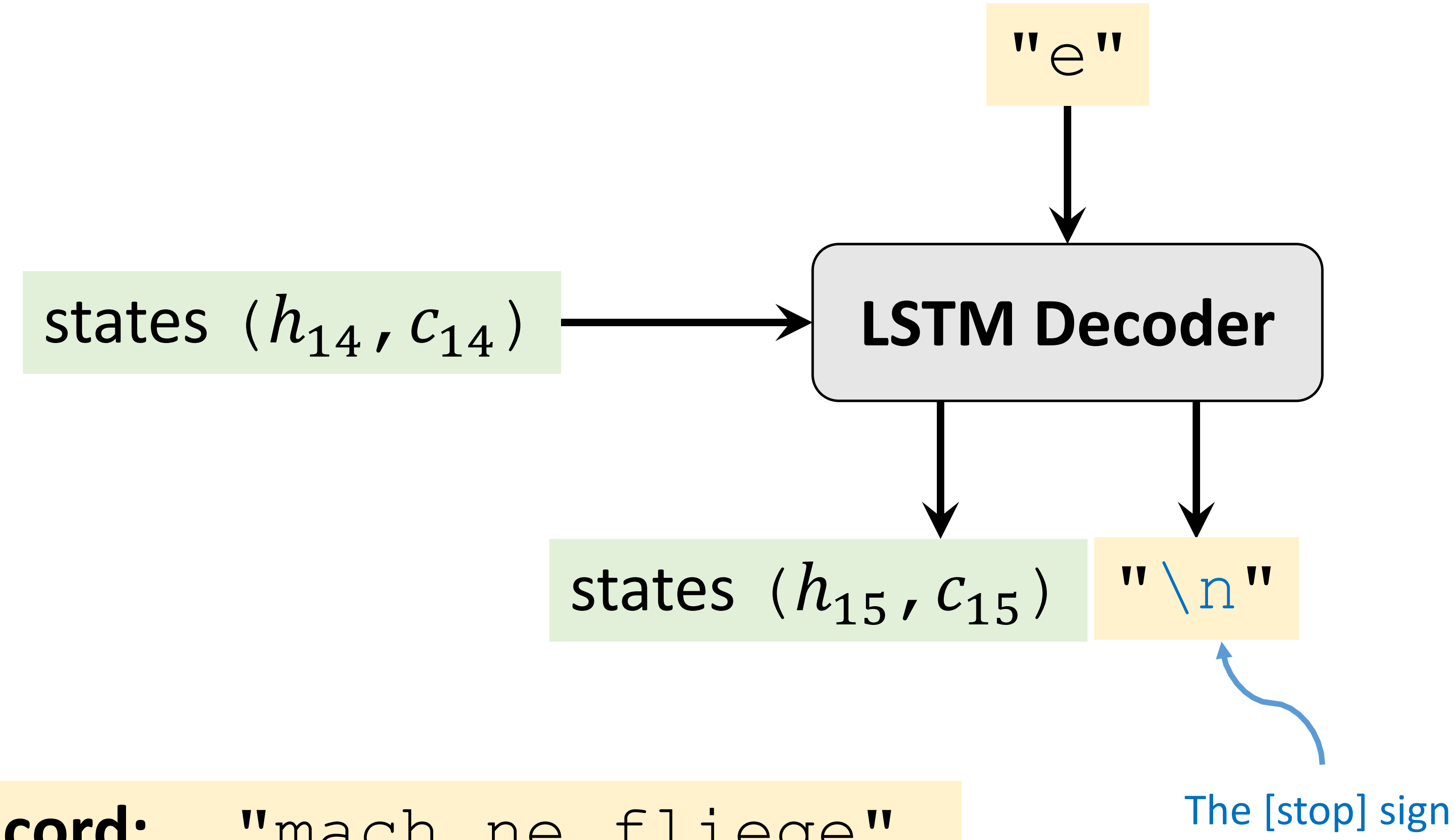

"a"
Repeat this process
states (h_1, c_1)
LSTM Decoder
states (h_3, c_3)
"c"
Record: "mac"

Inference



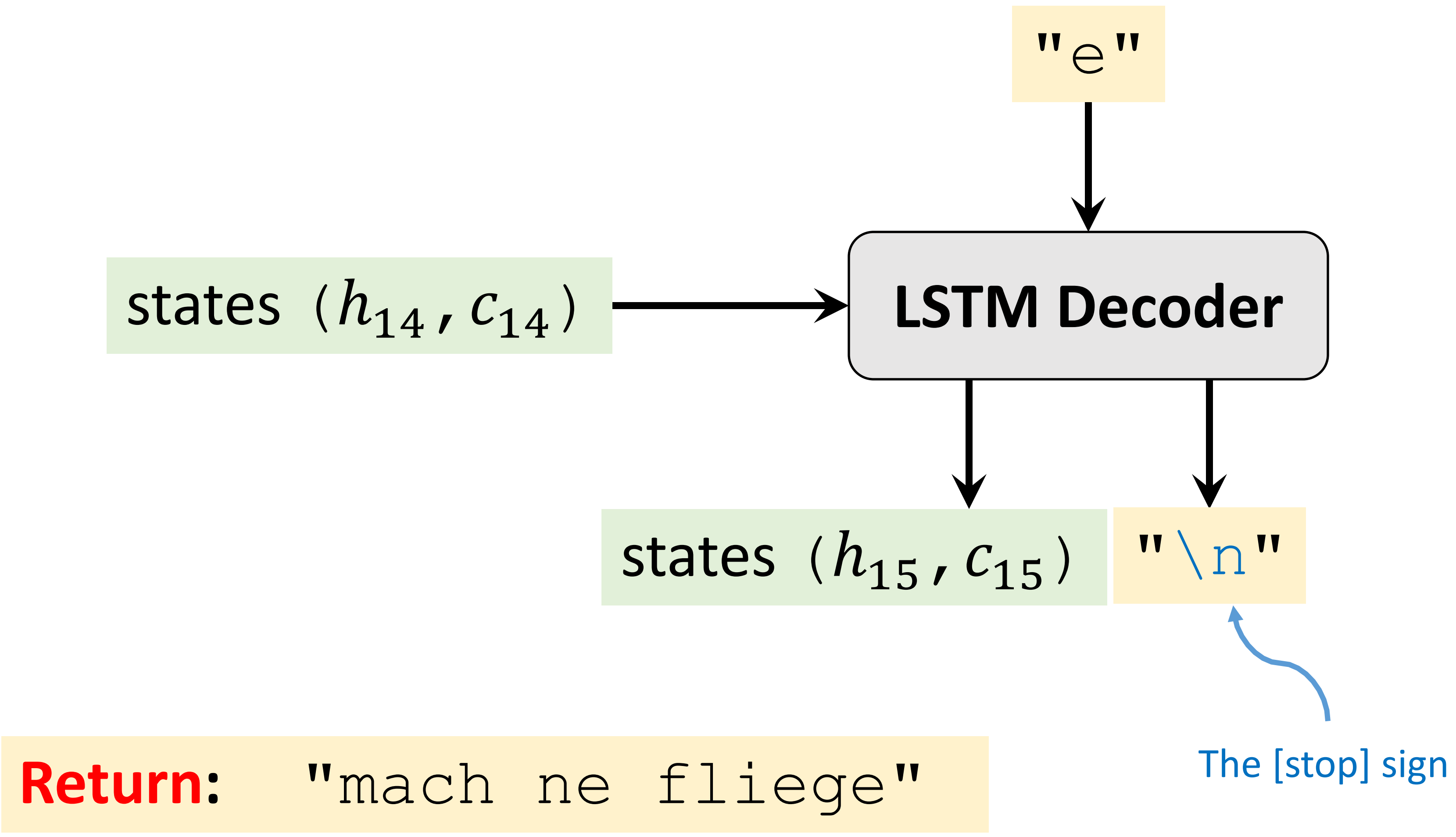
Record: `"mach ne fliege"`

Inference



Record: `"mach ne fliege"`

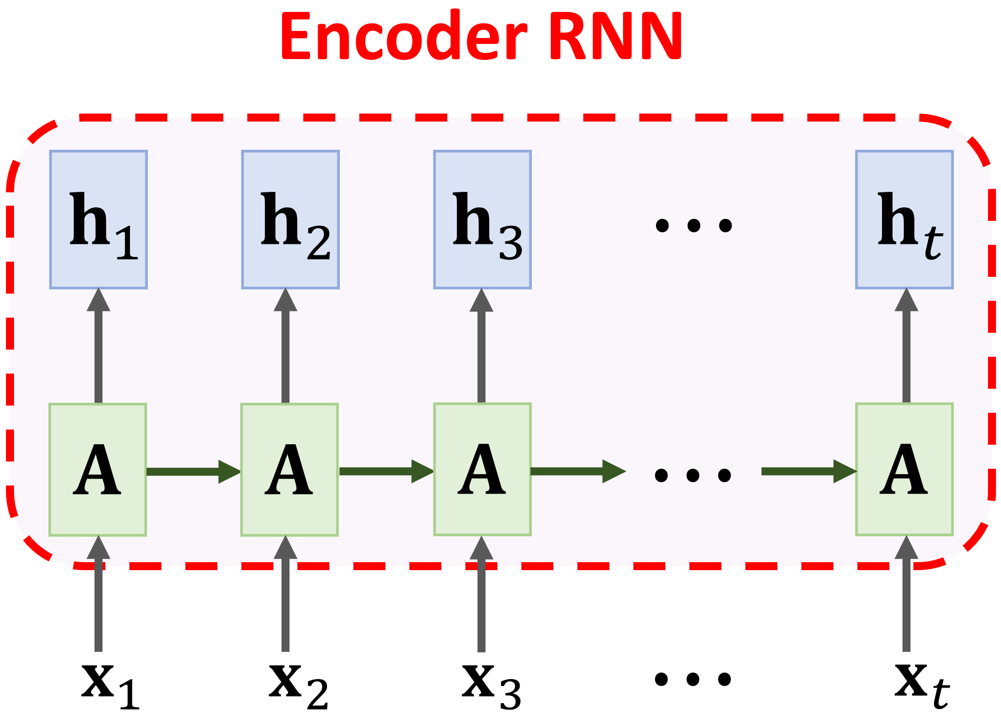
Inference



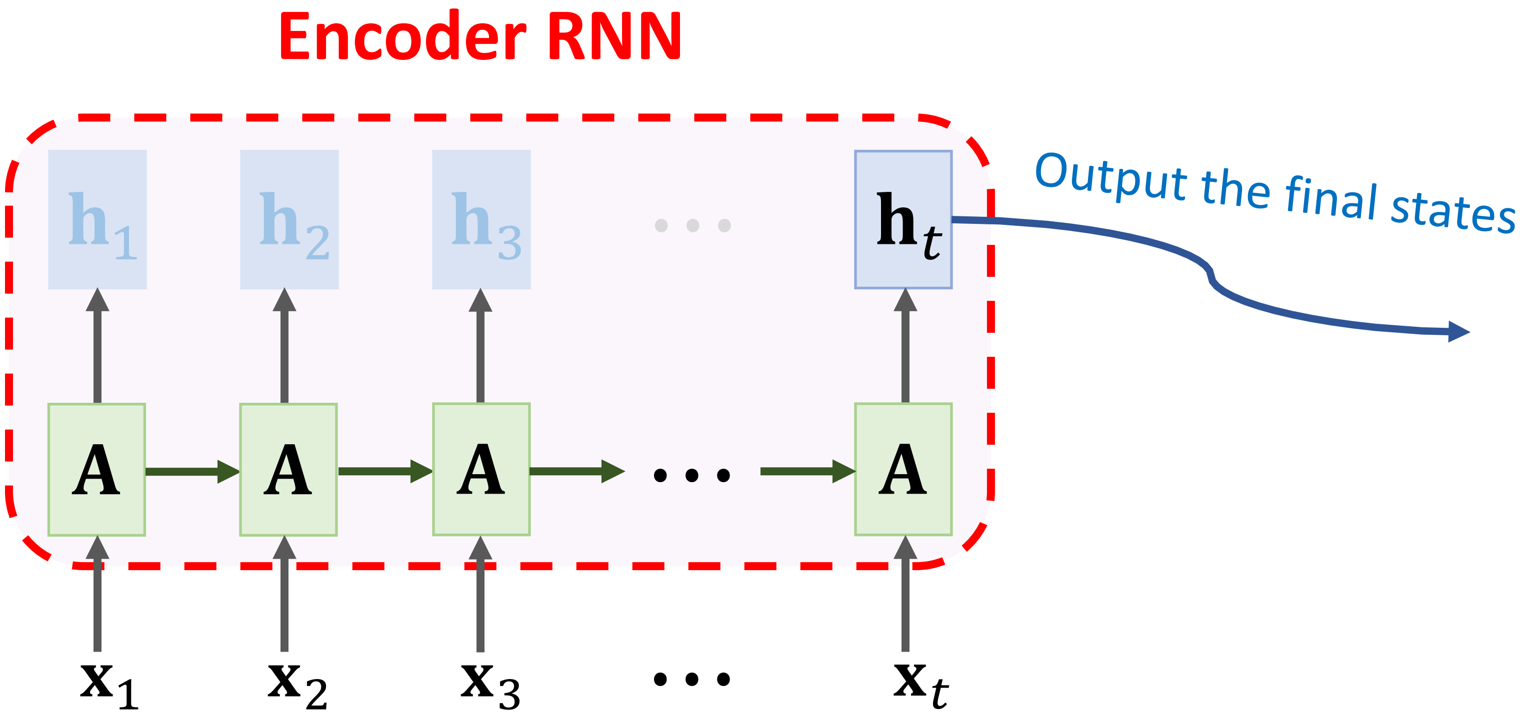
"e"
states (h_{14}, c_{14})
LSTM Decoder
states (h_{15}, c_{15})
"\n"
The [stop] sign
Return: "mach ne fliege"

Summary

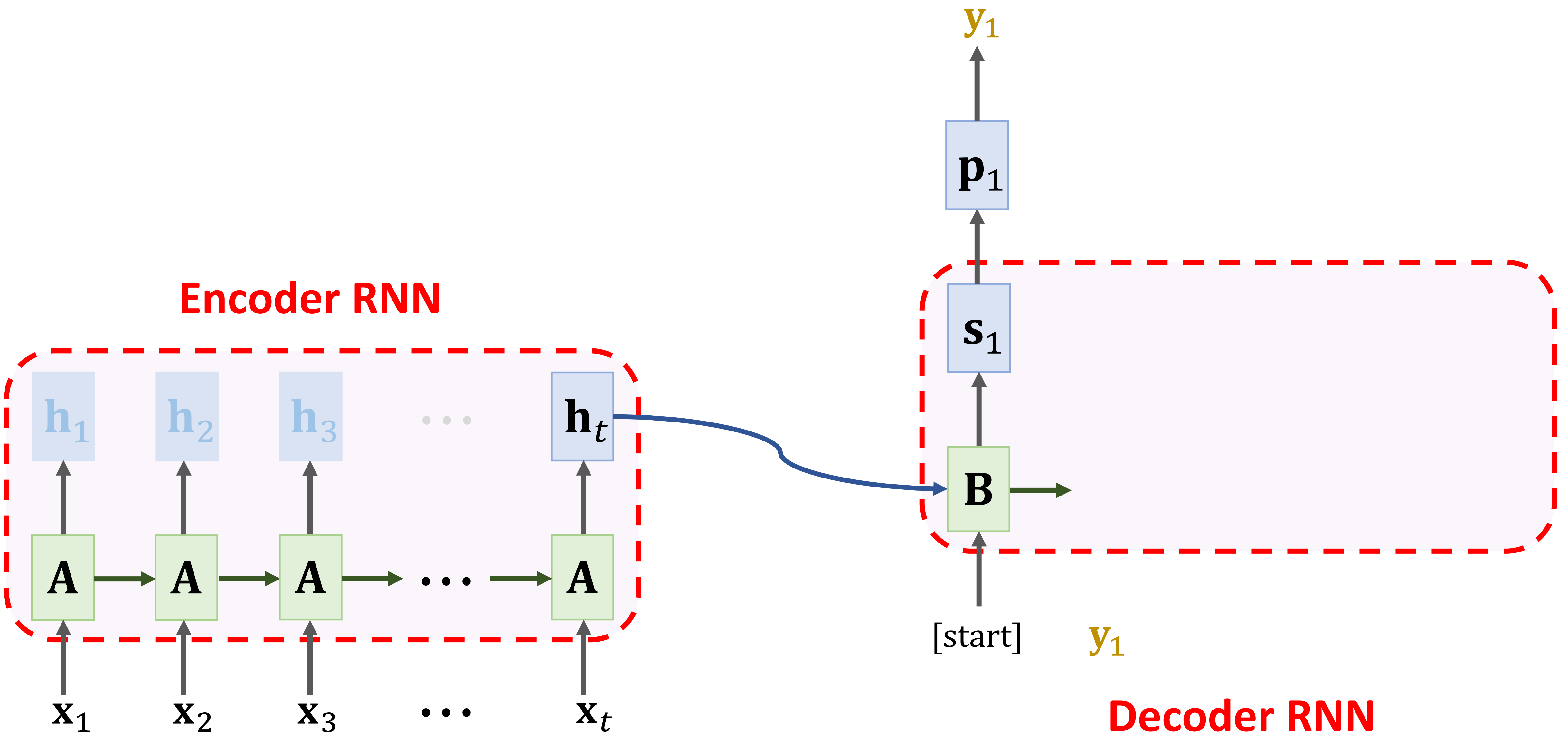
Seq2Seq Model



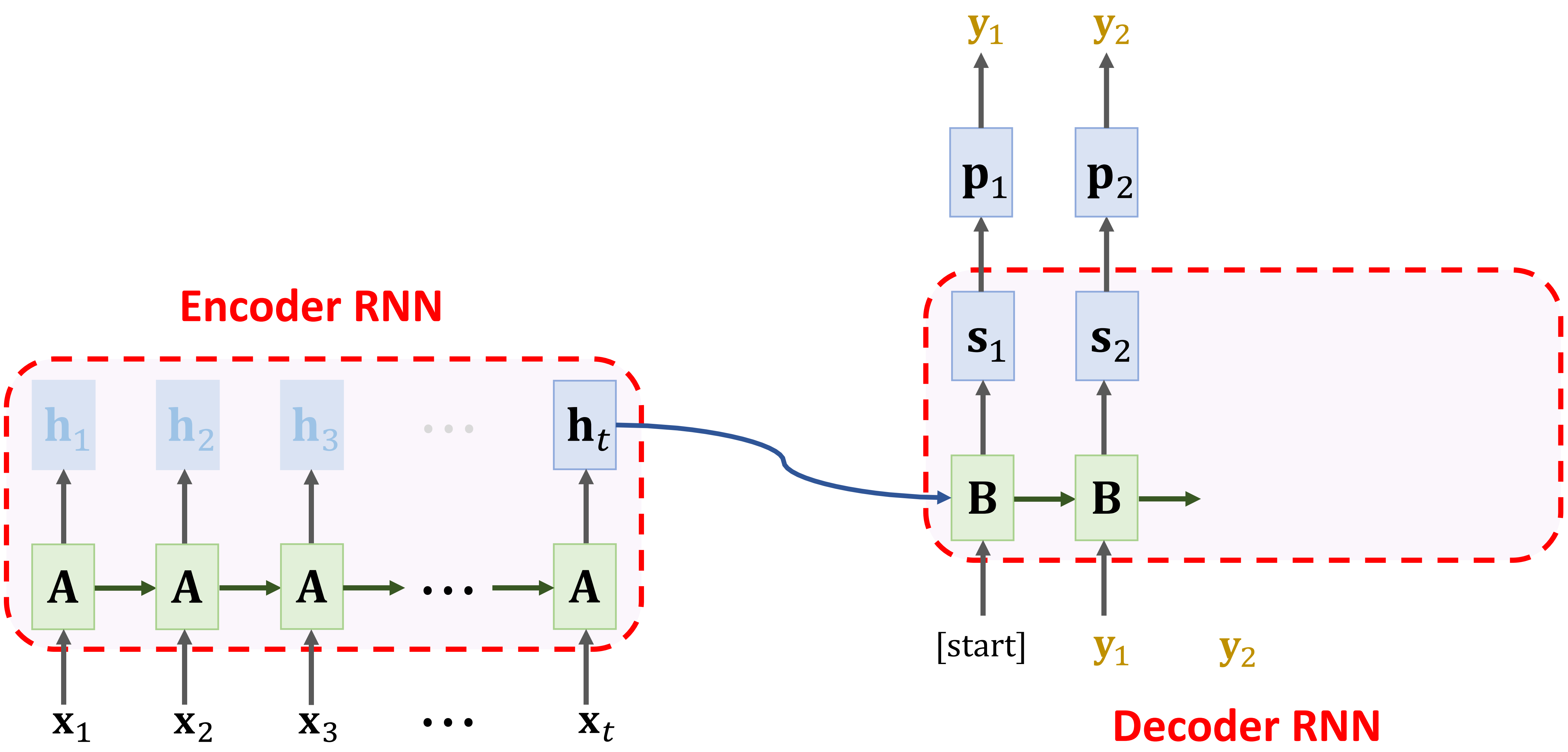
Seq2Seq Model



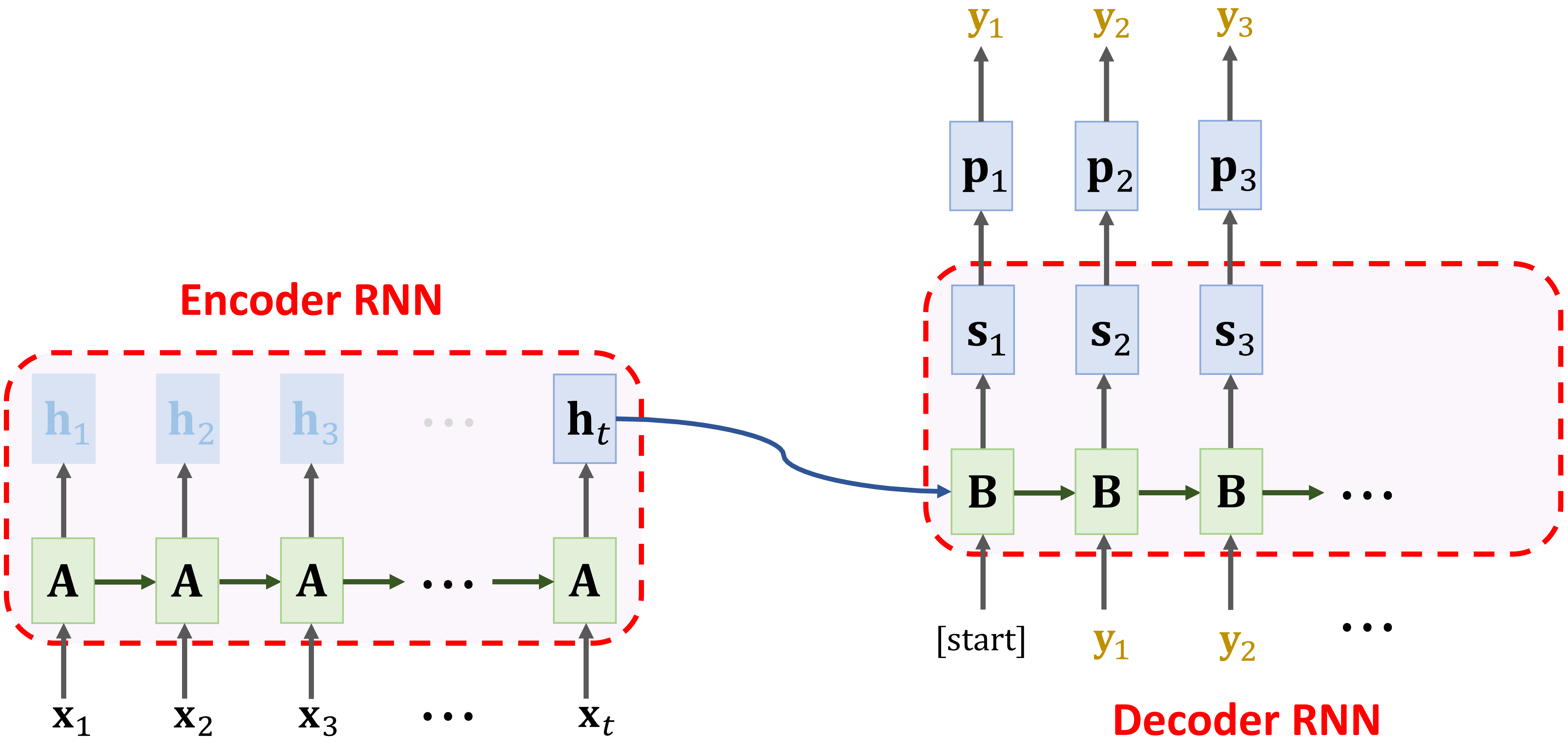
Seq2Seq Model



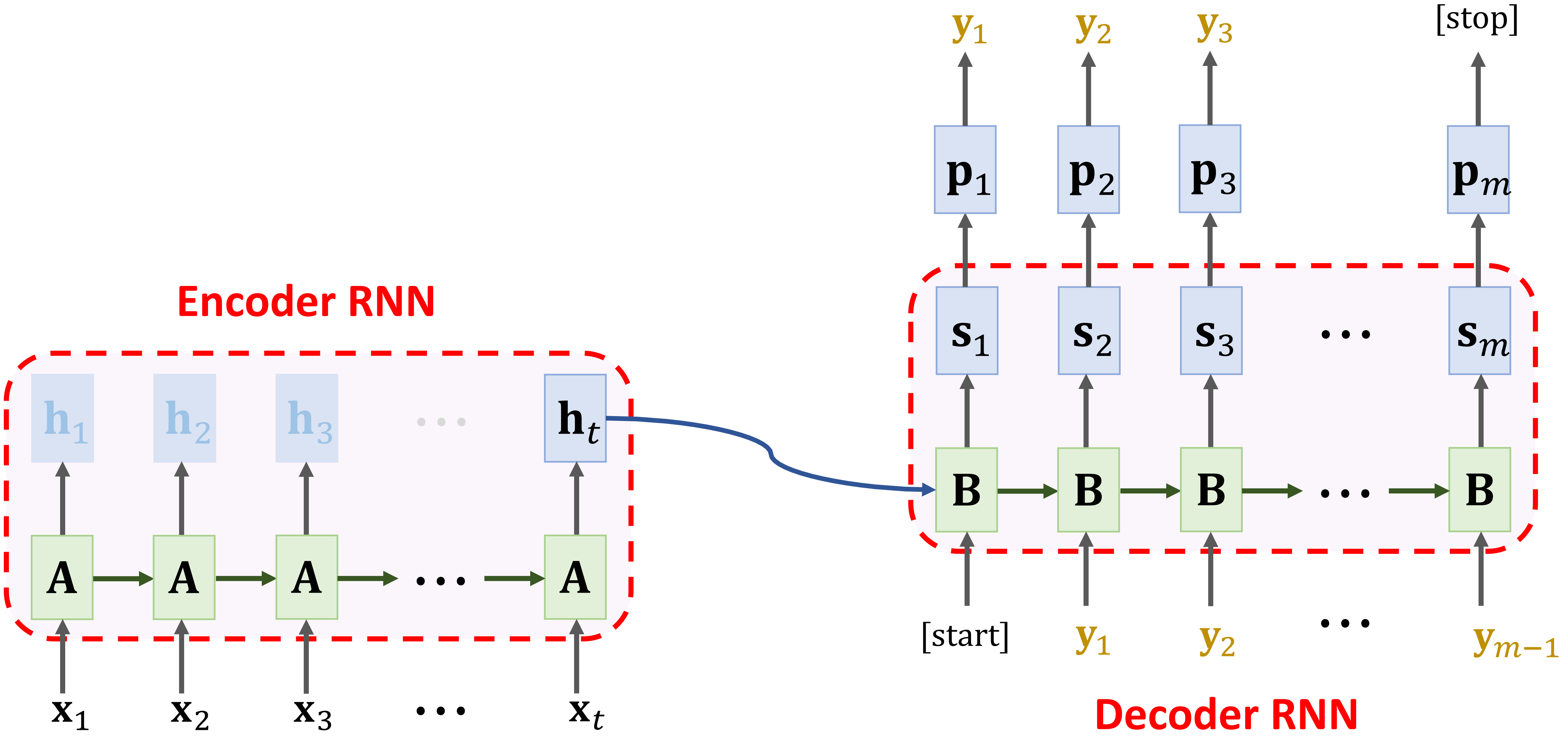
Seq2Seq Model



Seq2Seq Model



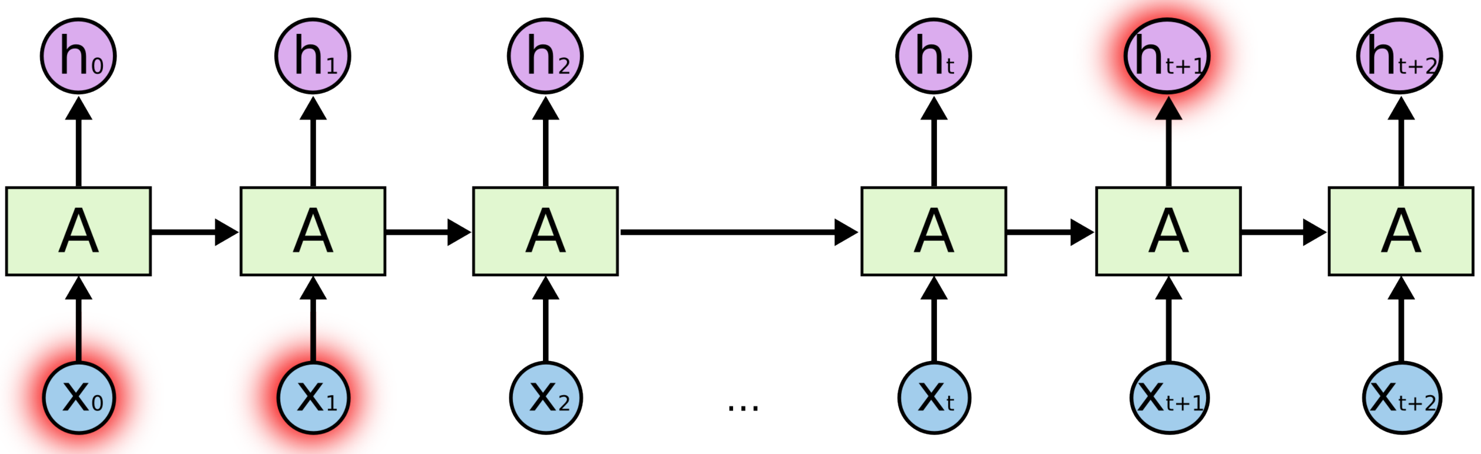
Seq2Seq Model



How to Improve?

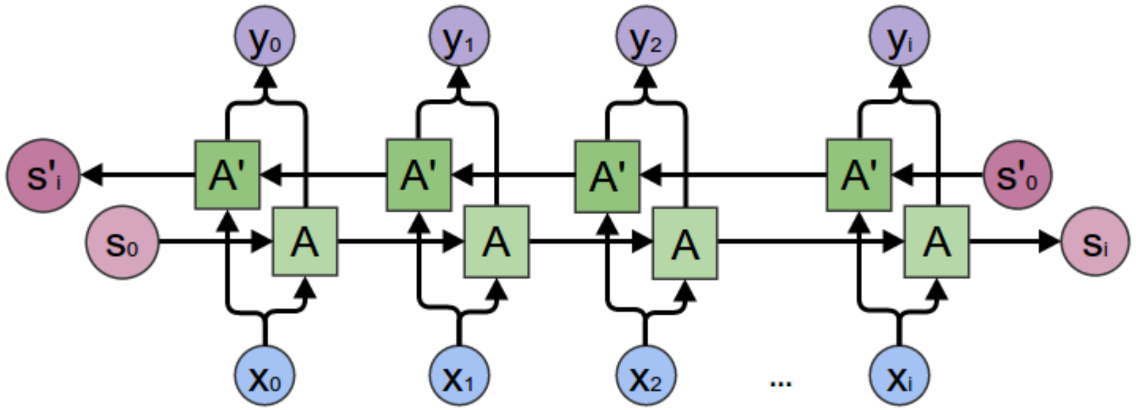
1. Bi-LSTM instead of LSTM

- The final states ($\mathbf{h}_t$ and $\mathbf{c}_t$) of the Encoder have all the information of the English sentence.
- Really?
  - If the sentence is long (hundreds of tokens), the final states have forgotten the first tokens.



1. Bi-LSTM instead of LSTM

- The final states ($\mathbf{h}_t$ and $\mathbf{c}_t$) of the Encoder have all the information of the English sentence.
- Really?
  - If the sentence is long (hundreds of tokens), the final states have forgotten the first tokens.
- Bi-LSTM (left-to-right and right-to-left) remembers the first tokens.



2. Word-Level Tokenization

- Word-level tokenization instead of char-level.
  - The average length of English words is 4.5 letters.
  - The sequences will be 4.5x shorter.
  - Shorter sequence ➔ less likely to forget.

2. Word-Level Tokenization

- Word-level tokenization instead of char-level.
  - The average length of English words is 4.5 letters.
  - The sequences will be 4.5x shorter.
  - Shorter sequence ➔ less likely to forget.
- But you will need a large dataset!
  - # of (frequently used) chars is ~10^2 ➔ one-hot suffices.
  - # of (frequently used) words is ~10^4 ➔ must use embedding.
  - Embedding Layer has many parameters ➔ overfitting!