

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/>

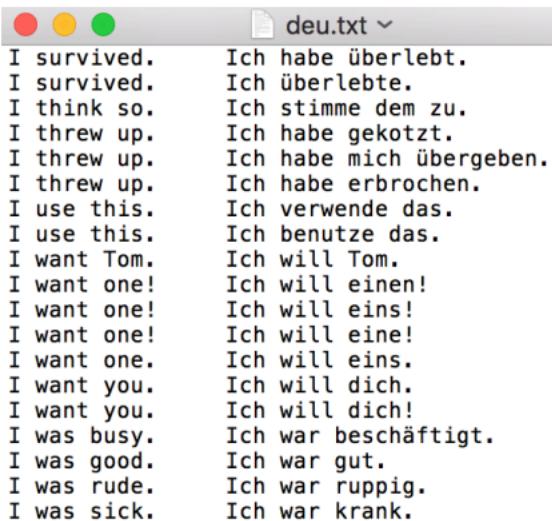
The screenshot shows a list of language pairs and their zip files. A red box highlights the German-English pair, and a red arrow points from this box to the right-hand table.

- Danish - English [dan-eng.zip](#) (14535)
- Dutch - English [nld-eng.zip](#) (28224)
- Estonian - English [est-eng.zip](#) (1550)
- Finnish - English [fin-eng.zip](#) (49208)
- French - English [fra-eng.zip](#) (155507)
- Galician - English [glg-eng.zip](#) (126)
- Georgian - English [kat-eng.zip](#) (125)
- German - English [deu-eng.zip](#) (170297)
- Greek - English [ell-eng.zip](#) (14485)
- Hebrew - English [heb-eng.zip](#) (126137)
- Hindi - English [hin-eng.zip](#) (2867)
- Hungarian - English [hun-eng.zip](#) (54197)
- Icelandic - English [isl-eng.zip](#) (6664)
- Indonesian - English [ind-eng.zip](#) (6702)
- Italian - English [ita-eng.zip](#) (297241)

	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.

Processing the Data

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



The screenshot shows a terminal window with three colored window control buttons (red, yellow, green) at the top. The title bar reads "deu.txt". The main area contains a list of 20 sentence pairs, each consisting of an English sentence on the left and a German translation on the right. The English sentences are mostly variations of "I [verb]". The German translations are their German equivalents. The text is in a monospaced font.

English Sentence	German Translation
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.

Processing the Data

- **Preprocessing:** to lower case, remove punctuation, remove non-printable chars.
- Get a list of English-German pairs.

```
for i in range(3000, 3010):
    print('[' + clean_pairs[i, 0] + ']' => '[' + clean_pairs[i, 1] + ']')

[my feet hurt] => [meine fue tun weh]
[my feet hurt] => [meine fue taten weh]
[my feet hurt] => [mir tun die fue weh]
[my hip hurts] => [mir tut die hufte weh]
[my jaw hurts] => [mir tut der kiefer weh]
[no one cares] => [niemanden interessiert]
[no one cares] => [niemand kummert sich darum]
[no one cares] => [interessiert niemanden]
[no one knows] => [das wei niemand]
[nobody asked] => [niemand fragte]
```

Processing the Data

- **input_texts** : a list of English sentences.
- **target_texts**: a list of German sentences (begin with '\t' and end with '\n').

```
input_texts = clean_pairs[:, 0]
target_texts = ['\t' + text + '\n' for text in clean_pairs[:, 1]]
```

```
input_texts[3000]
```

```
'my feet hurt'
```

```
target_texts[3000]
```

```
'\tmeine fue tun weh\n'
```

Tokenize the Texts in the Character Level

- input_texts => [**Eng_Tokenizer**] => encoder_input_seq
- target_texts => [**Deu_Tokenizer**] => decoder_input_seq



Use 2 different tokenizers for the 2 languages

Tokenize the Texts in the Character Level

- `input_texts` => [**Eng_Tokenizer**] => `encoder_input_seq`
- `target_texts` => [**Deu_Tokenizer**] => `decoder_input_seq`

```
input_texts[100]
```

'go away'

```
encoder_input_seq[100]
```

```
array([17, 4, 1, 7, 16, 7, 15, 0, 0, 0, 0, 0, 0],
```

dtype=int32)

g' o' _' a' ... 'y'

0, 0, 0, 0, 0, 0, 0]

Zero-padding: make all the sequences the same length.

Tokenize the Texts in the Character Level

- `input_texts` => [**Eng_Tokenizer**] => `encoder_input_seq`
 - `target_texts` => [**Deu_Tokenizer**] => `decoder_input_seq`

```
target_texts[100]
```

'\tmach ne fliege\n'

decoder input seq[100]

```
array([ 8, 13, 10, 12, 7, 1, 4, 2, 1, 21, 18, 3, 2, 17, 2, 9, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      dtype=int32)
't'   'm'   'a'   ...   ...   'g'   'e'   '\n'
```

Tokenize the Texts in the Character Level

- `input_texts` => [**Eng_Tokenizer**] => `encoder_input_seq`
- `target_texts` => [**Deu_Tokenizer**] => `decoder_input_seq`

28: 26 letters, blank, and zero-padding

- 15-dim vector
- 15×28 matrix
- One-hot encode the sequences:
 - `encoder_input_seq` => [**one_hot**] => `encoder_input_data`
 - `decoder_input_seq` => [**one_hot**] => `decoder_input_data`

51-dim vector

51×30 matrix

30: 26 letters, '\t', '\n', blank, and zero-padding

Summary of the Data Processing

- Raw data: lists of English and German sentences.

Summary of the Data Processing

- Raw data: lists of English and German sentences.

- Sequences:
 - encoder_input_seq: 10000×15
 - decoder_input_seq: 10000×51
- number of sentences max length of English sentences
-
- max length of German sentences

Summary of the Data Processing

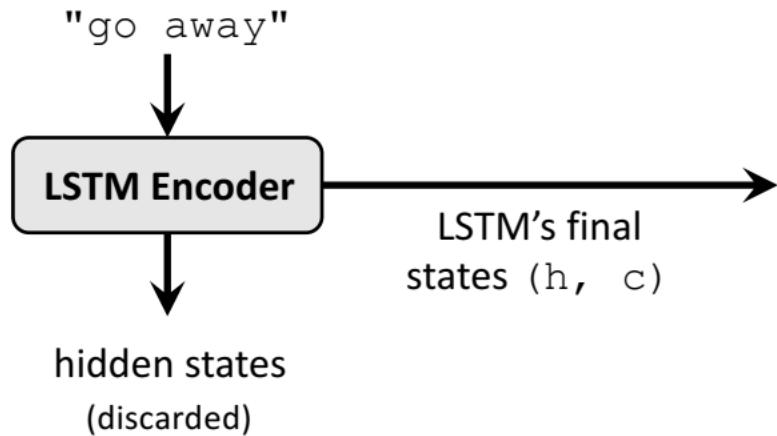
- Raw data: lists of English and German sentences.

- Sequences:
 - encoder_input_seq: 10000×15
 - decoder_input_seq: 10000×51
- number of sentences** **max length of English sentences**
- max length of German sentences**

- One-hot encode:
 - encoder_input_data: $10000 \times 15 \times 28$
 - decoder_input_data: $10000 \times 51 \times 30$
- #unique chars**

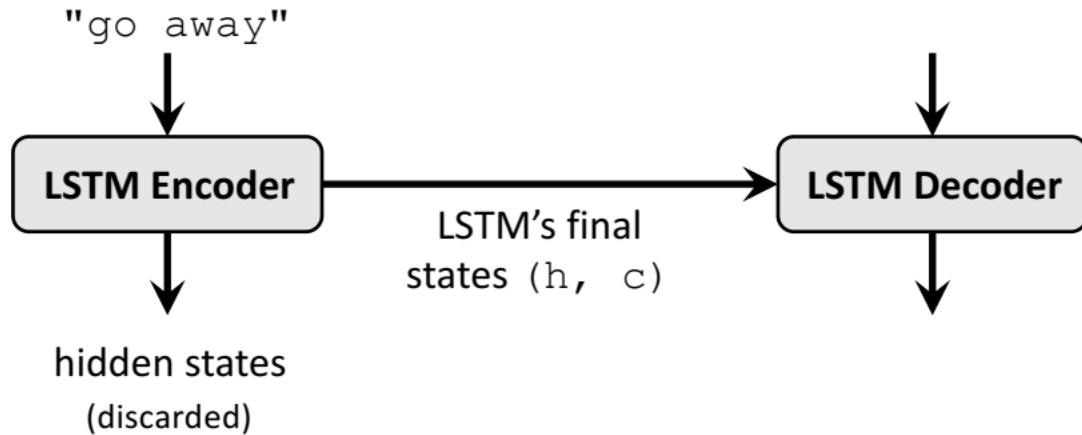
Seq2Seq Model

Training of the Seq2Seq Model

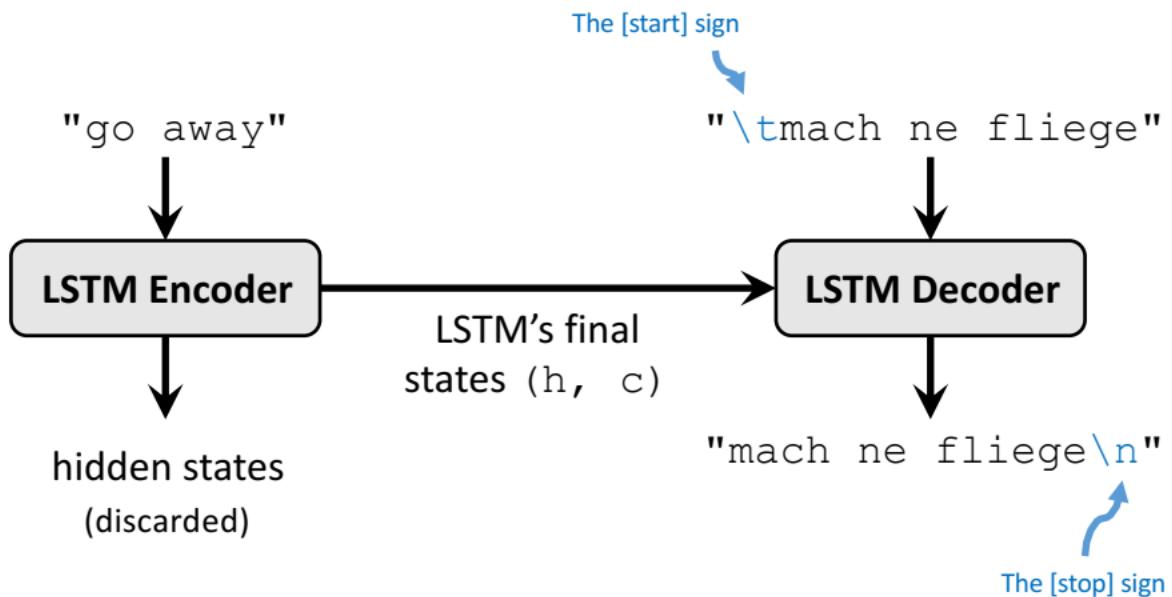


Training of the Seq2Seq Model

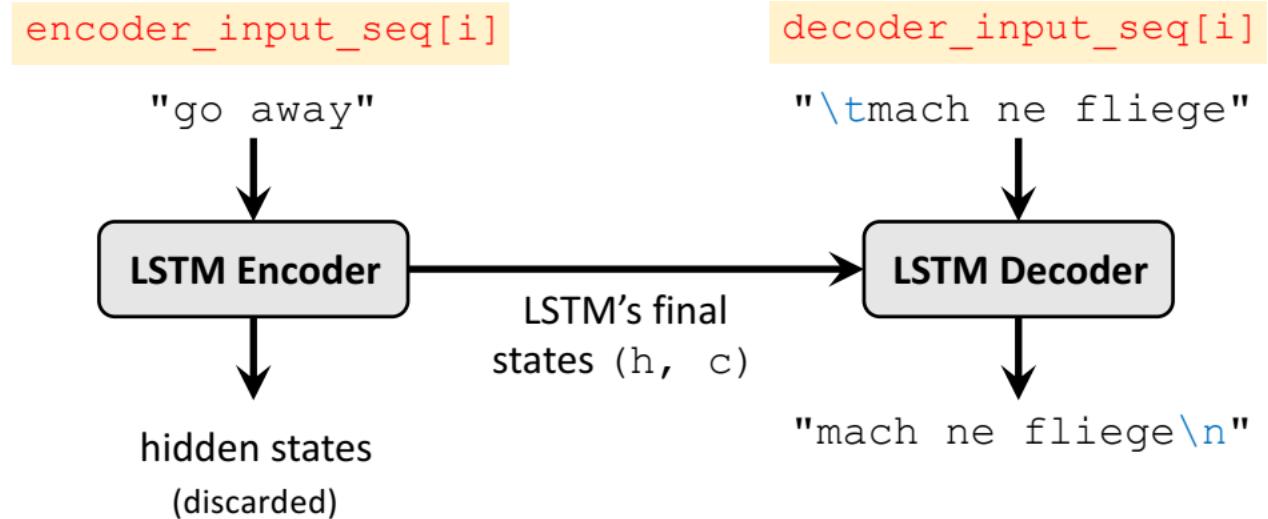
The [start] sign



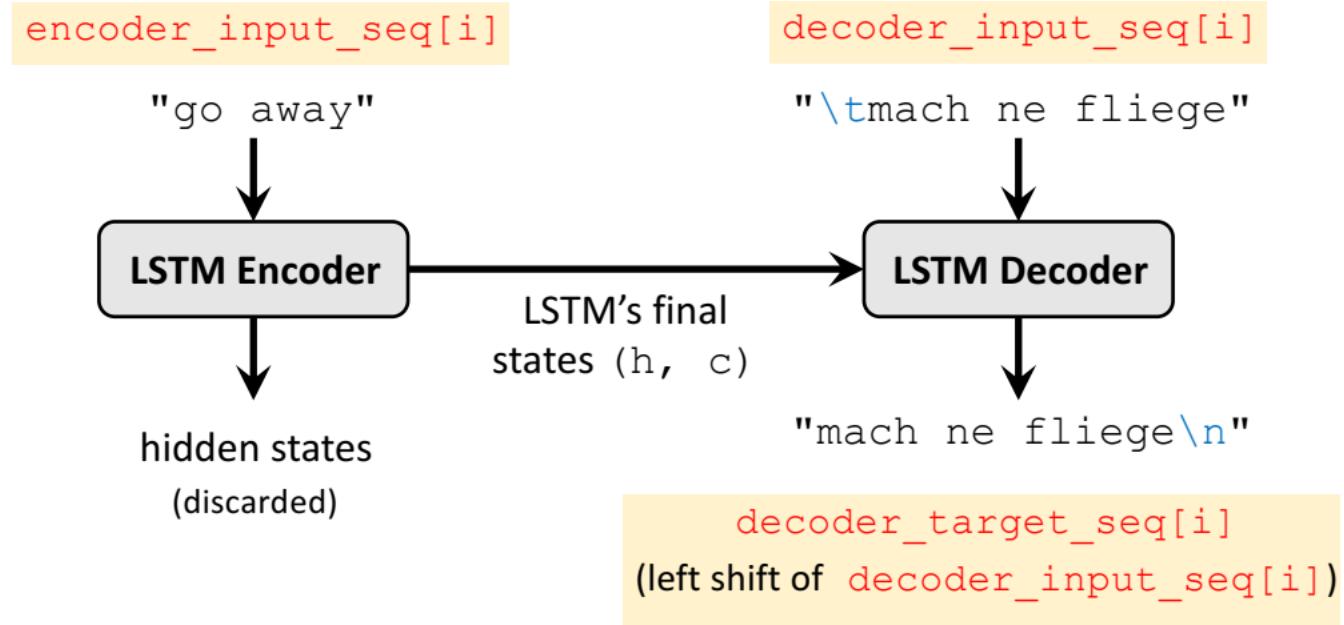
Training of the Seq2Seq Model



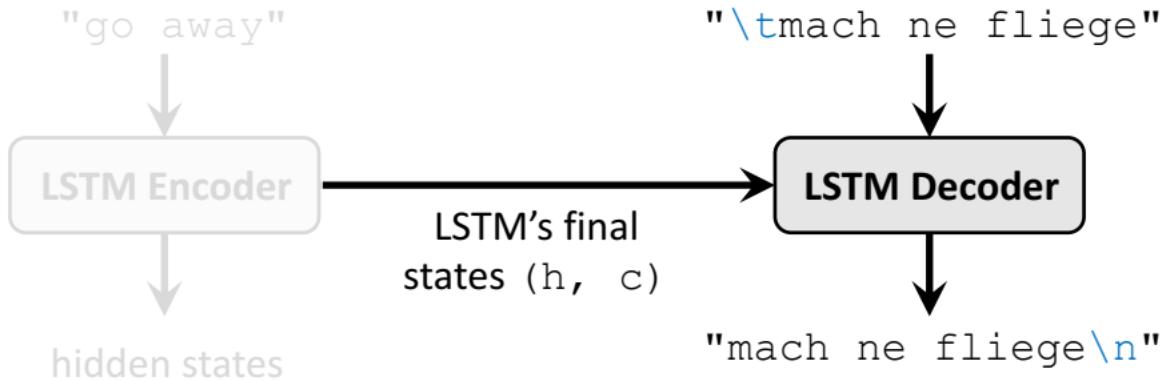
Training of the Seq2Seq Model



Training of the Seq2Seq Model



Training of the Seq2Seq Model

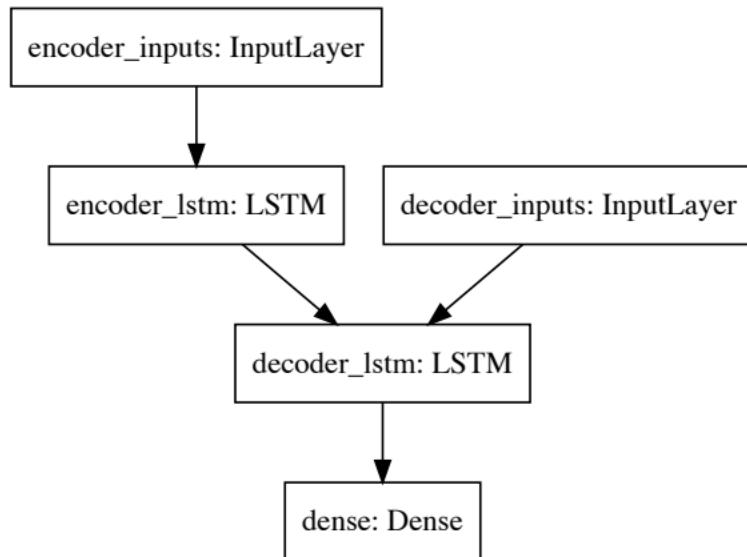


(dashed arrow)

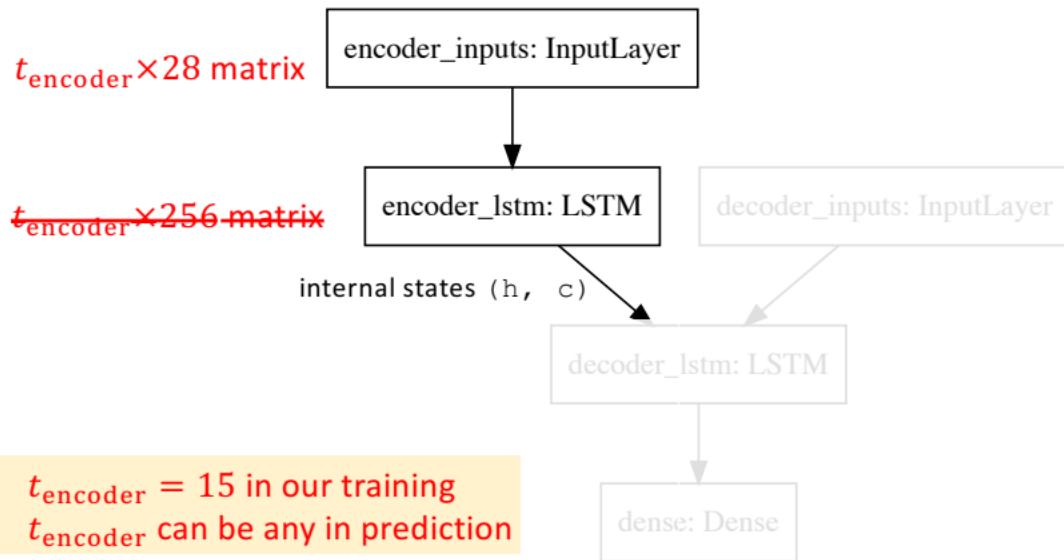
- The decoder is a text generator (in the previous lecture).
- Difference from the simple text generator: the initial states are determined by the encoder.

Seq2Seq Model in Keras

Implementation Using Keras



Encoder Network



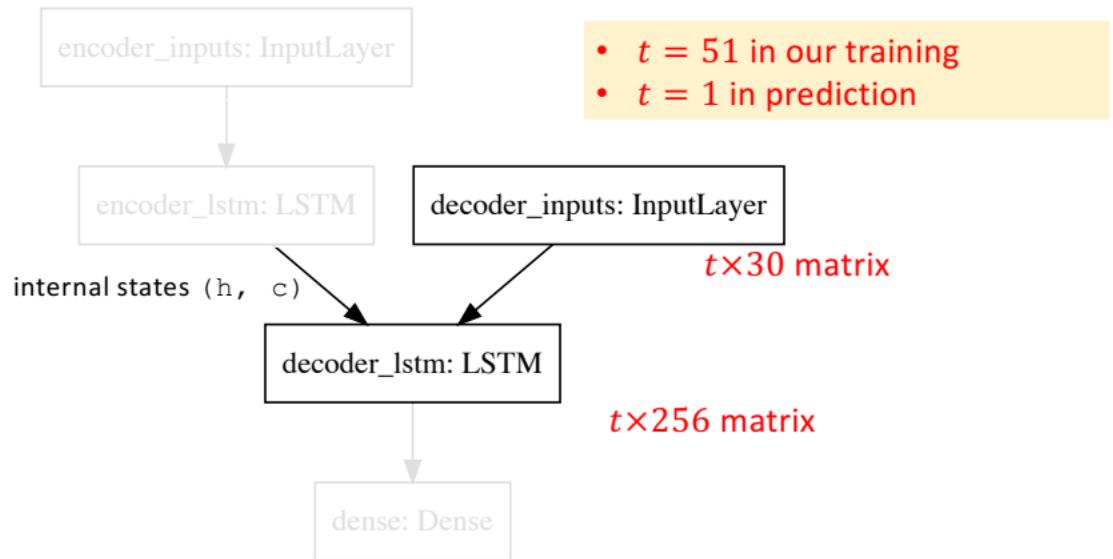
Encoder Network

```
from keras.layers import Input, LSTM

latent_dim = 256
encoder_inputs = Input(shape=(None, num_encoder_tokens), name='encoder_inputs')

encoder_lstm = LSTM(latent_dim, return_state=True,
                    dropout=0.5, name='encoder_lstm')
state_h, state_c = encoder_lstm(encoder_inputs)
discarded
encoder_states = [state_h, state_c]
```

Decoder Network

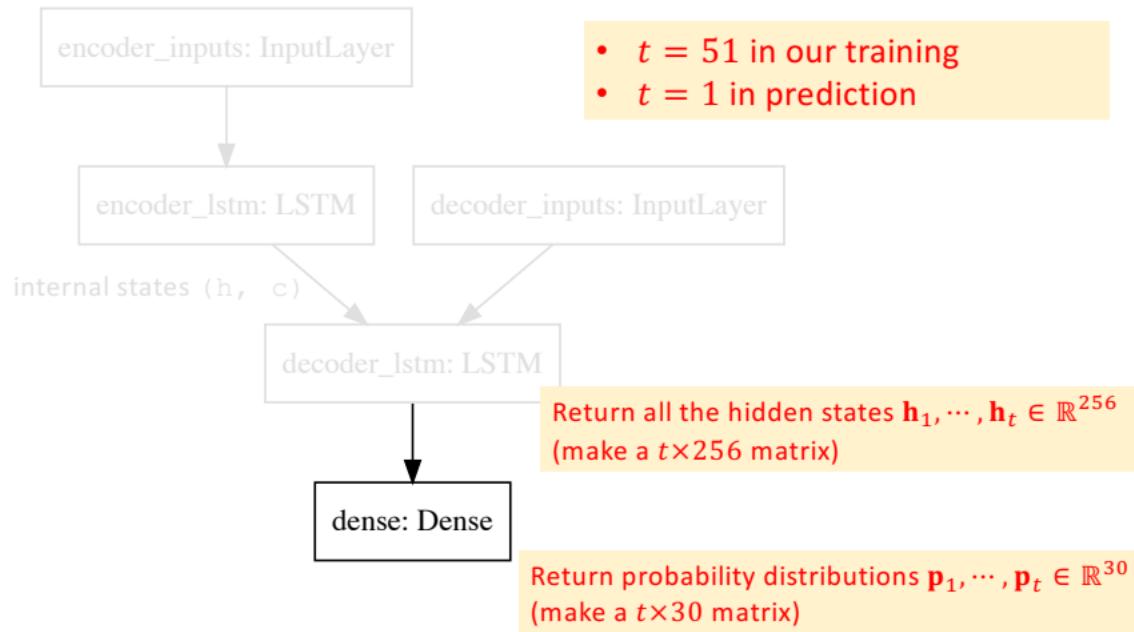


Decoder Network

```
from keras.layers import Input, LSTM

latent_dim = 256
decoder_inputs = Input(shape=(None, num_decoder_tokens), name='decoder_inputs')
decoder_lstm = LSTM(latent_dim, return_sequences=True,
                    Return  $h_t$  and  $c_t$  return_state=True, dropout=0.5, name='decoder_lstm')
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
From the encoder network
```

The Dense Output Layer (Softmax Classifier)



The Dense Output Layer (Softmax Classifier)

```
from keras.layers import Input, LSTM, Dense
from keras.models import Model

latent_dim = 256

decoder_inputs = Input(shape=(None, num_decoder_tokens), name='decoder_inputs')

decoder_lstm = LSTM(latent_dim, return_sequences=True,
                    return_state=True, dropout=0.5, name='decoder_lstm')
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
    num_decoder_tokens = 30 in our experiment
decoder_dense = Dense(num_decoder_tokens, activation='softmax', name='dense')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

The Dense Output Layer (Softmax Classifier)

```
from keras.layers import Input, LSTM, Dense
from keras.models import Model

latent_dim = 256

decoder_inputs = Input(shape=(None, num_decoder_tokens), name='decoder_inputs')

decoder_lstm = LSTM(latent_dim, return_sequences=True,
                    return_state=True, dropout=0.5, name='decoder_lstm')
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)

decoder_dense = Dense(num_decoder_tokens, activation='softmax', name='dense')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Connect the Whole Network

```
from keras.layers import Input, LSTM, Dense
from keras.models import Model

latent_dim = 256

decoder_inputs = Input(shape=(None, num_decoder_tokens), name='decoder_inputs')
    One-hot encode of a German sentence.

decoder_lstm = LSTM(latent_dim, return_sequences=True,
                    return_state=True, dropout=0.5, name='decoder_lstm')
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)

decoder_dense = Dense(num_decoder_tokens, activation='softmax', name='dense')
decoder_outputs = decoder_dense(decoder_outputs)
    Probability distributions. (Predictions for the next chars.)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

One-hot encode of an English sentence.

Training

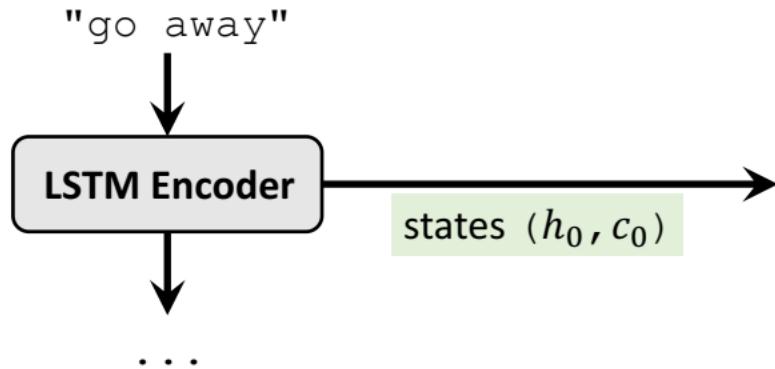
```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=64, epochs=50, validation_split=0.2)

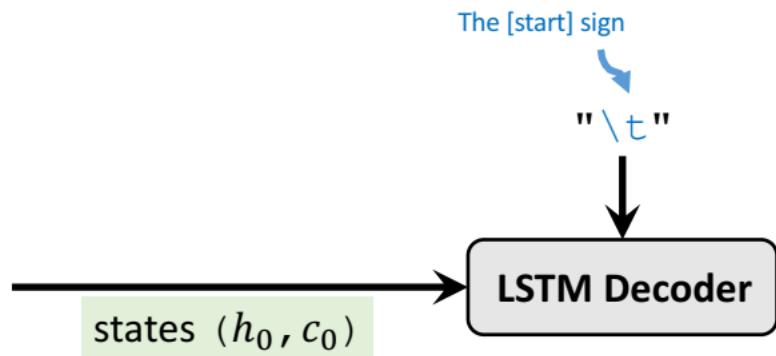
Train on 8000 samples, validate on 2000 samples
Epoch 1/50
8000/8000 [=====] - 17s 2ms/step - loss: 1.2445 - val_loss: 1.2276
Epoch 2/50
8000/8000 [=====] - 18s 2ms/step - loss: 0.9616 - val_loss: 0.9512
  •
Epoch 49/50
8000/8000 [=====] - 22s 3ms/step - loss: 0.3637 - val_loss: 0.4842
Epoch 50/50
8000/8000 [=====] - 22s 3ms/step - loss: 0.3609 - val_loss: 0.4806
```

Inference Using the Seq2Seq Model

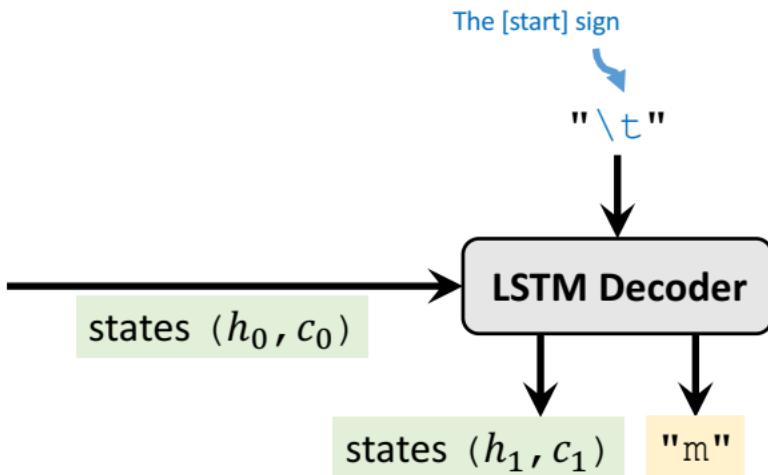
Inference



Inference

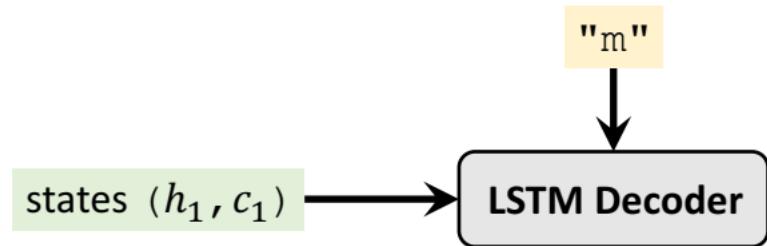


Inference



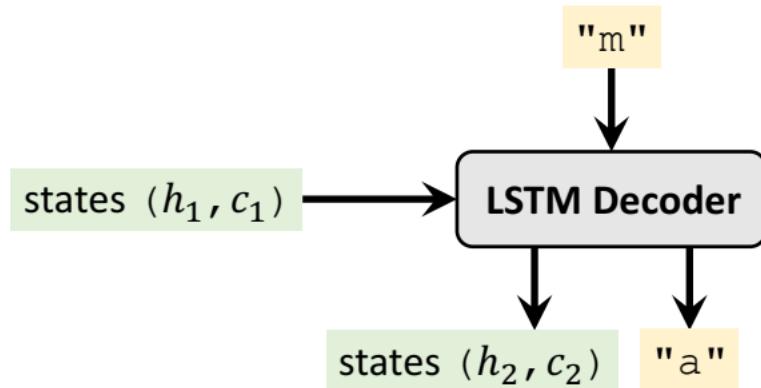
Record: "m"

Inference



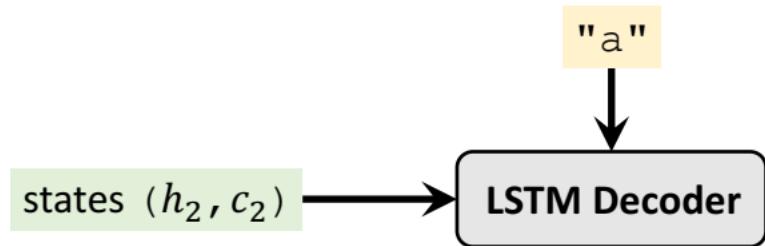
Record: "m"

Inference



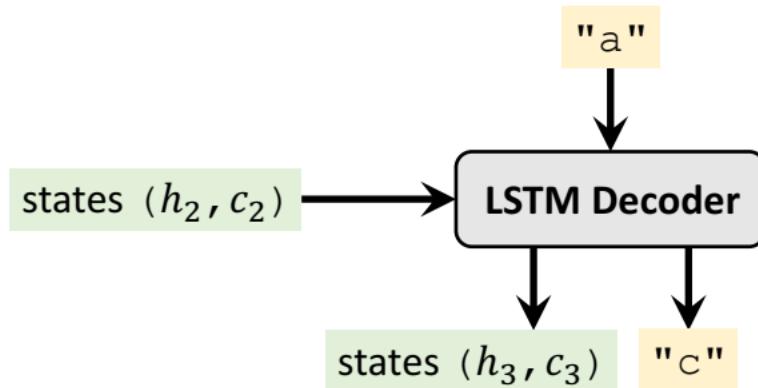
Record: "ma"

Inference



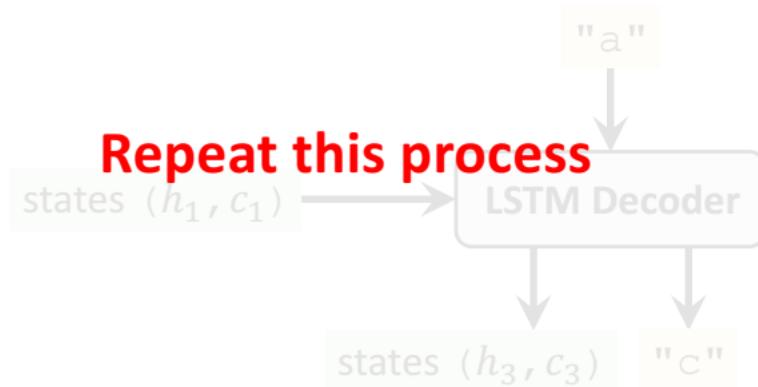
Record: "ma"

Inference



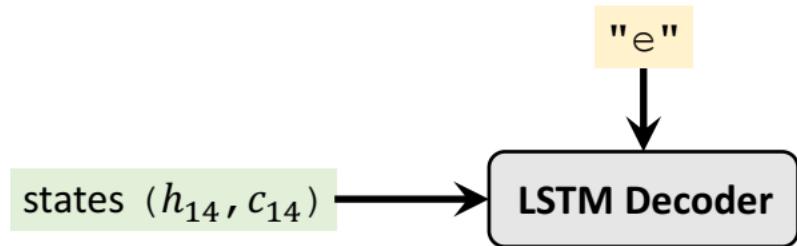
Record: "mac"

Inference



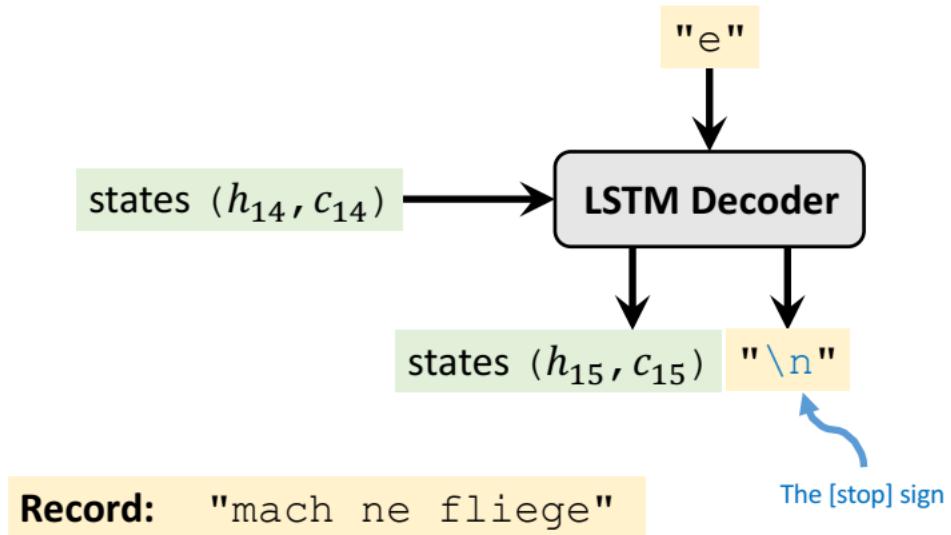
Record: "mac"

Inference

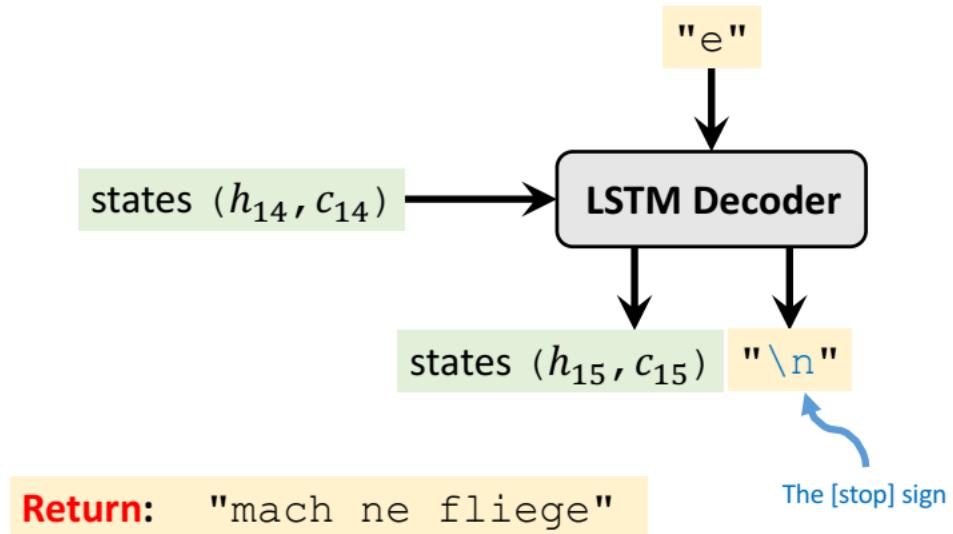


Record: "mach ne fliege"

Inference



Inference



Examples

-
English: go away
German (true): scher dich weg
German (pred): verpiss dich

-
English: go away
German (true): scher dich fort
German (pred): verpiss dich

-
English: go away
German (true): geh weg
German (pred): verpiss dich

-
English: go away
German (true): verpiss dich
German (pred): verpiss dich

-
English: wait up
German (true): warten sie mal
German (pred): warten sie auf

-
English: wait up
German (true): wartet mal
German (pred): warten sie auf

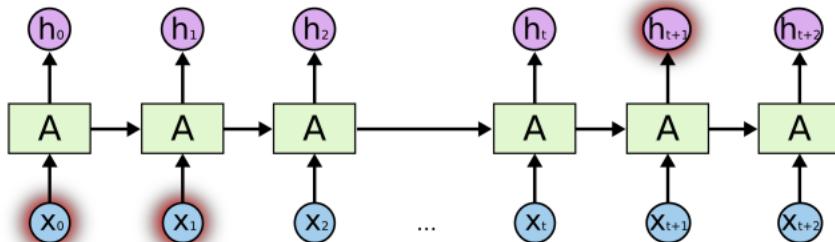
-
English: wake up
German (true): wach auf
German (pred): wachen sie auf

-
English: wake up
German (true): wachen sie auf
German (pred): wachen sie auf

How to Improve?

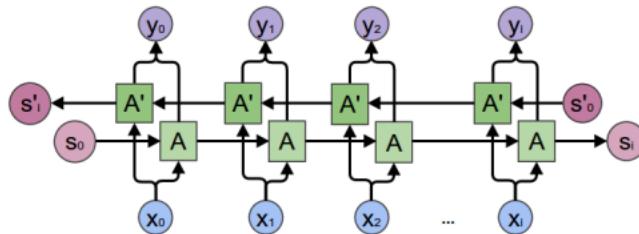
1. Bi-LSTM instead of LSTM

- 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.
- But you will need a large dataset!
 - # of (frequently used) chars is $\sim 10^2$.
 - # of (frequently used) words is $\sim 10^4$.
 - Word Embedding Layer has many parameters → overfitting!

3. Multi-Task Learning

- Even if you want to **translate English to German**, you can use all the datasets:

-  Afrikaans - English [afr-eng.zip](#) (725)
-  Aklanon - English [akl-eng.zip](#) (22)
-  Albanian - English [sqi-eng.zip](#) (412)
-  Algerian Arabic - English [arq-eng.zip](#) (156)
-  Arabic - English [ara-eng.zip](#) (11009)
-  Arabic (Gulf) - English [afb-eng.zip](#) (28)
-  Assamese - English [asm-eng.zip](#) (23)
-  Asturian - English [ast-eng.zip](#) (23)
-  Azerbaijani - English [aze-eng.zip](#) (2131)
-  Basque - English [eus-eng.zip](#) (667)
-  Belarusian - English [bel-eng.zip](#) (2698)
-  Bengali - English [ben-eng.zip](#) (4399)
-  Berber - English [ber-eng.zip](#) (54988)
-  Bulgarian - English [bul-eng.zip](#) (14968)

• • •

-  Spanish - English [spa-eng.zip](#) (120799)
-  Swedish - English [swe-eng.zip](#) (17409)
-  Tagalog - English [tgl-eng.zip](#) (3144)
-  Tamil - English [tam-eng.zip](#) (197)
-  Tatar - English [tat-eng.zip](#) (529)
-  Telugu - English [tel-eng.zip](#) (138)
-  Thai - English [tha-eng.zip](#) (110)
-  Turkish - English [tur-eng.zip](#) (497249)
-  Ukrainian - English [ukr-eng.zip](#) (113396)
-  Urdu - English [urd-eng.zip](#) (1180)
-  Uyghur - English [uig-eng.zip](#) (285)
-  Vietnamese - English [vie-eng.zip](#) (3413)
-  Waray - English [war-eng.zip](#) (1181)
-  Zaza - English [zza-eng.zip](#) (345)

3. Multi-Task Learning

- Even if you want to **translate English to German**, you can use all the datasets.
- [English ==> German], [English ==> French], [English ==> Spanish] ...
 - The same **Encoder Network**.
 - Different **Decoder Network**.
 - Build 100 seq2seq models sharing the **Encoder Network**.
 - Train the 100 seq2seq models on the 100 datasets.
- 100x more datasets → better trained **Encoder Network**.

How to Improve?

1. Bi-LSTM instead of LSTM.
2. Tokenization in the word-level (rather than char-level.)
3. Multi-task learning.
4. Attention! (Next lecture.)

Homework

- Build a seq2seq model for machine translation.
 - Anything languages except for [English ==> German].
 - Data processing, tokenization, build networks, training, and prediction.
 - Follow my IPython Notebook.
- Make as least one improvement over my naïve model.
 - E.g., Bi-LSTM, multi-task, attention, etc.
- Evaluate your model using BLEU score. (Optional.)
 - BLEU (BiLingual Evaluation Understudy).
 - Reference:
 - Wikipedia: <https://en.wikipedia.org/wiki/BLEU>
 - Blog: <https://machinelearningmastery.com/calculate-bleu-score-for-text-python/>