Introduction to sequence to sequence models

NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

Sequence to sequence generation

- Output a sequence given a sequence as input.
- Fixed length input.
- Fixed length output.
- Input output length different in general.

Seq2seq applications

- Machine translation.
- Question answering.
- NER/POS-Tagging.
- Text summarization.
- Grammar correction.

Text summarization

Input:

```
"Russian Defense Minister Ivanov called Sunday for the creation of a
joint front for combating global terrorism."
```

Output:

```
"Russia calls for joint front against terrorism."
```

Grammar correction

Input:

```
"There is no a doubt, tracking systems has brought many benefits in this
information age."
```

Output:

```
"There is no doubt, tracking systems have brought many benefits in this
information age."
```

English French dataset

```
I know.    Je sais.
I left.    Je suis parti.
I'm OK.    Je vais bien.
Got it!    J'ai pigé !
Really?    Vraiment??
Shut up!    Taisez-vous?!
Have fun.    Amuse-toi bien !
```

[1] http://www.manythings.org/anki/

Preprocess ENG-FRA dataset

```
# Split i-th line into two at the tab character
eng_fra_line = str(lines[i]).split('\t')
```

```
# Separate out the English sentence
eng_line = eng_fra_line[0]
```

```
# Append start and end token to French sentence
fra_line = '\t' + eng_fra_line[1] + '\n'
```

```
# Append the English and French sentence to the list of sentences
english_sentences.append(eng_line)
french_sentences.append(fra_line)
```

English vocabulary

```
# Create an empty set to contain the English vocabulary
english_vocab = set()
```

```
# Iterate over each English sentence
for eng_line in english_sentences:
    # Iterate over each character of each sentence
    for ch in eng_line:
        # Add the character to the vocabulary if it is already not there
        if (ch not in english_vocab):
            english_vocab.add(ch)
```

```
# Sort the vocabulary
english_vocab = sorted(list(english_vocab))
```

French vocabulary

```
# Create an empty set to contain the French vocabulary
french_vocab = set()
```

```
# Iterate over each French sentence
for fra_line in french_sentences:
    # Iterate over each character of each sentence
    for ch in fra_line:
        # Add the character to the vocabulary if it is already not there
        if (ch not in french_vocab):
            french_vocab.add(ch)
```

```
# Sort the vocabulary
french_vocab = sorted(list(french_vocab))
```

Mappings for English vocabulary

- Character to integer mapping for English vocabulary.

```
eng_char_to_idx = dict((char, idx) for idx, char
                                   in enumerate(english_vocab))
```

- Integer to character mapping for the English vocabulary.

```
eng_idx_to_char = dict((idx, char) for idx, char
                                   in enumerate(english_vocab))
```

Mappings for French vocabulary

- Character to integer mapping for French vocabulary.

```
fra_char_to_idx = dict((char, idx) for idx, char
                                  in enumerate(french_vocab))
```

- Integer to character mapping for the French vocabulary.

```
fra_idx_to_char = dict((idx, char) for idx, char
                                 in enumerate(french_vocab))
```

Let's practice!

NATURAL LANGUAGE GENERATION IN PYTHON

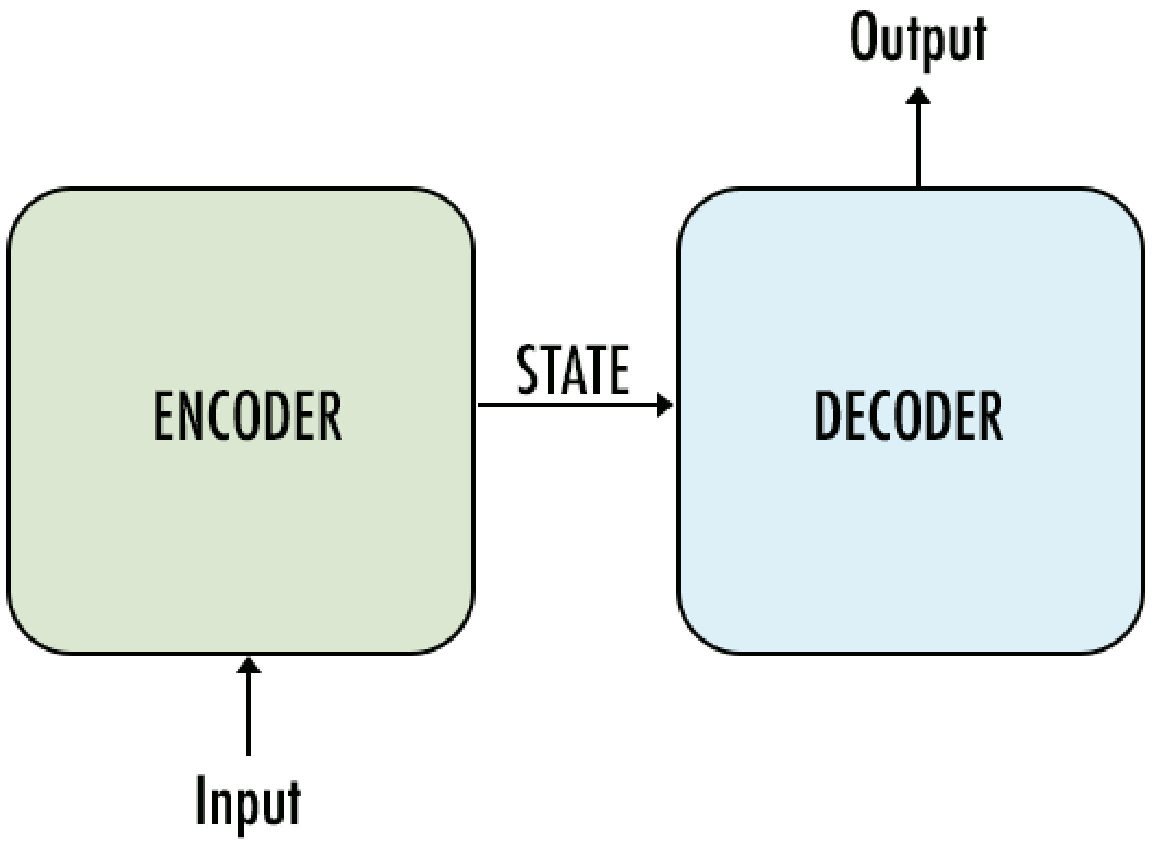
Neural machine translation

NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

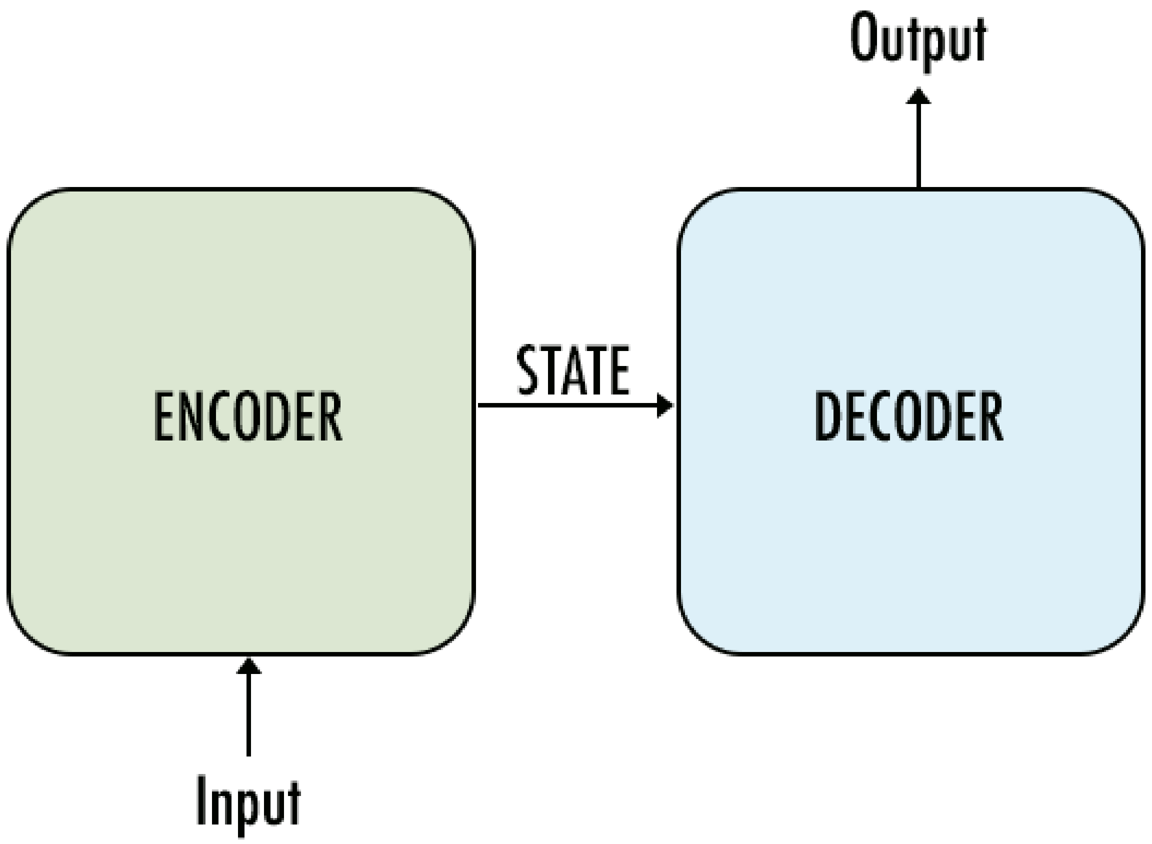
Encoder

- Accepts input sequence.
- Summarizes information in state vectors.
- State vectors passed to decoder.
- Outputs ignored.



Decoder

- Initial state vectors from encoder.
- Final states ignored.
- Outputs the predicted sequence.

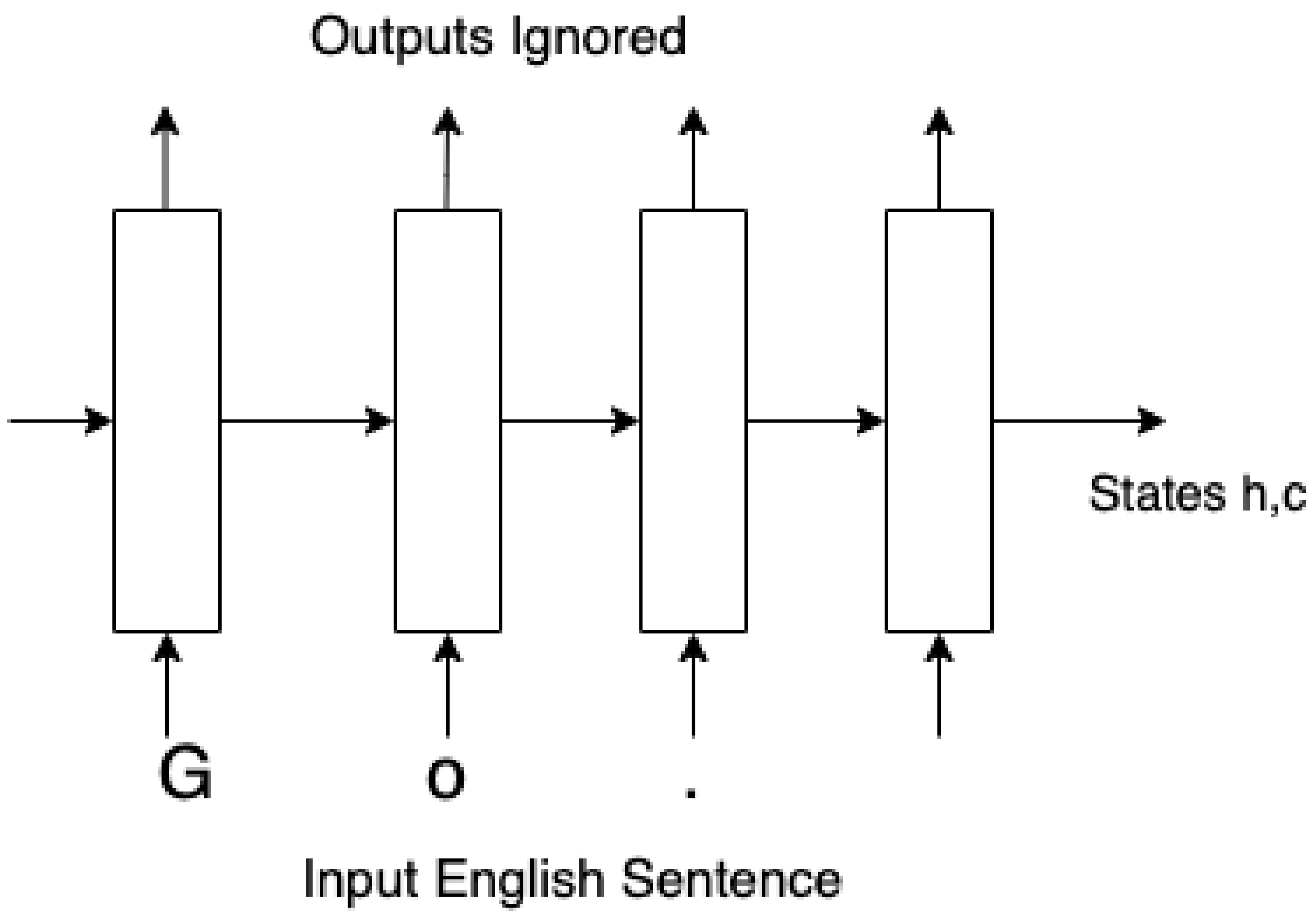


Teacher forcing

- Inference
  - Behavior as usual.
  - Input at each step - output from previous time step.
- Training
  - Input is actual output for the current step.
  - Not the predicted output from previous time step.
  - Known as teacher-forcing.

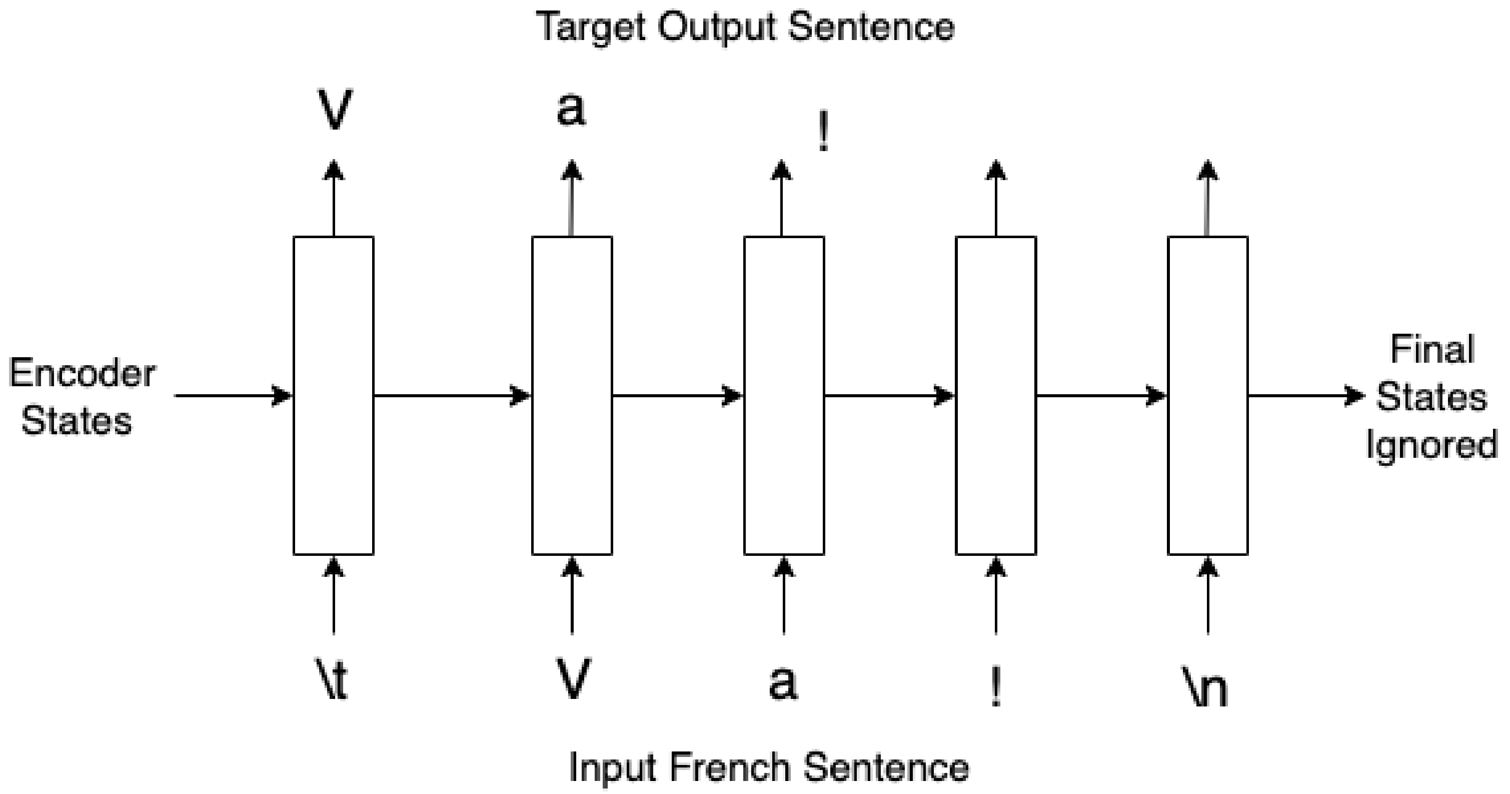
Encoder for translation

- Inputs : English sentences.
- Number of time steps : length of the sentence.
- States summarize the English sentences.
- Final states passed to decoder.
- Outputs ignored.

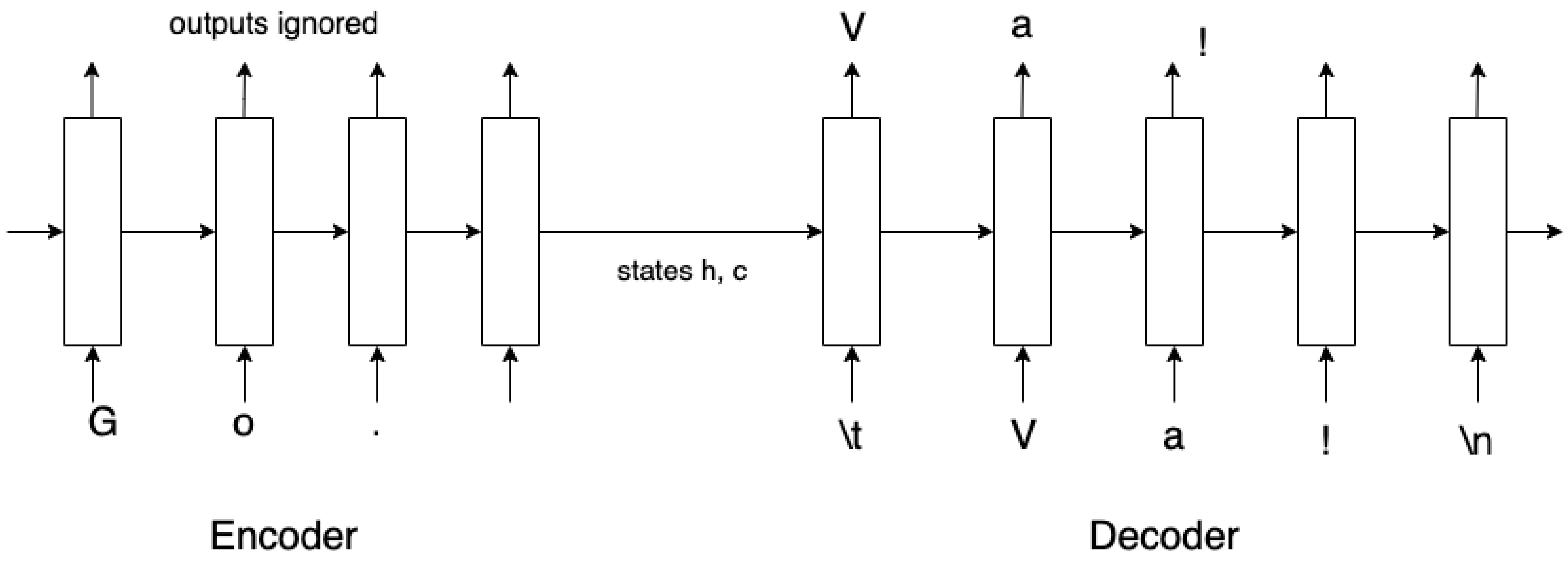


Decoder for translation

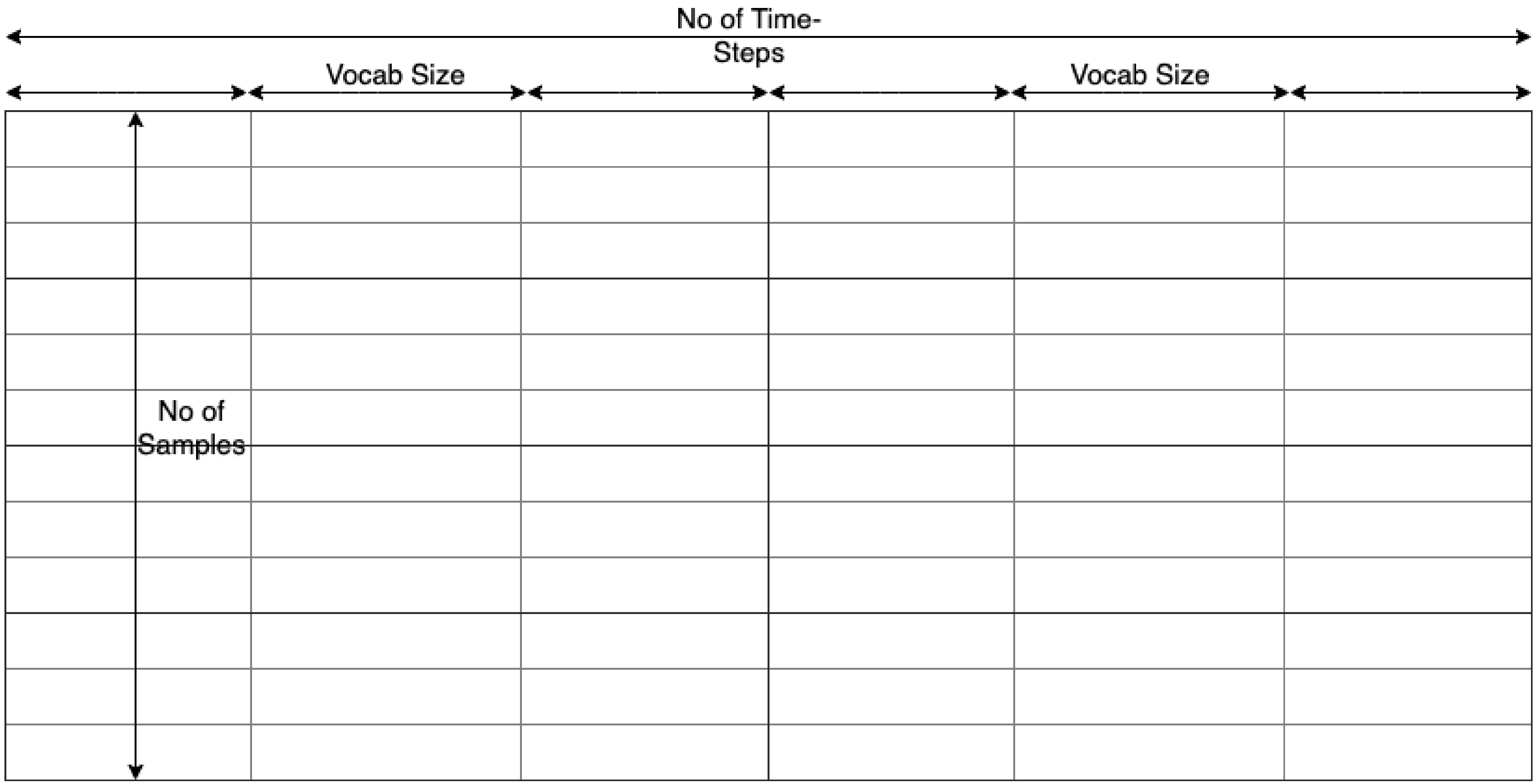
- Initial states : final states of the encoder.
- Inputs : French sentences.
- Outputs : translated sentences.
- Final states ignored.
- No of time-steps : length of French sentence.



Encoder-decoder during training



Shape of input and target vectors



Define input and target vectors

- Find the step sizes.

```
max_len_eng_sent = max([len(sentence) for sentence in english_sentences])
max_len_fra_sent = max([len(sentence) for sentence in french_sentences])
```

- Define the input and target vectors.

```
eng_input_data = np.zeros((len(english_sentences), max_len_eng_sent,
                           len(english_vocab)), dtype='float32')
fra_input_data = np.zeros((len(french_sentences), max_len_fra_sent,
                           len(french_vocab)), dtype='float32')
target_data = np.zeros((len(french_sentences), max_len_fra_sent,
                           len(french_vocab)), dtype='float32')
```

Initialize input and target vectors

```
for i in range(no_of_sentences):

    # Iterate over each character of English sentences
    for k, ch in enumerate(english_sentences[i]):
        eng_input_data[i, k, eng_char_to_idx[ch]] = 1.

    # Iterate over each character of French sentences
    for k, ch in enumerate(french_sentences[i]):
        fra_input_data[i, k, fra_char_to_idx[ch]] = 1.

        # Target data will be one timestep ahead
        if k > 0:
            target_data[i, k-1, fra_char_to_idx[ch]] = 1.
```

Keras functional APIs

```
# This returns a input vector of size 784
inputs = Input(shape=(784,))
```

```
# A dense layer of 64 units is called on a vector returning a tensor
predictions = Dense(64, activation='relu')(inputs)
```

```
# This creates a model with an Input layer and an output of a dense layer
model = Model(inputs=inputs, outputs=predictions)
```

Build the encoder

- Create input layer followed by the LSTM layer of 256 units.

```
encoder_input = Input(shape = (None, len(english_vocab)))
encoder_LSTM = LSTM(256, return_state = True)
```

- Feed input to the LSTM layer and get output.

```
encoder_outputs, encoder_h, encoder_c = encoder_LSTM(encoder_input)
```

- Ignore the output and save the states.

```
encoder_states = [encoder_h, encoder_c]
```

Build the decoder

- Create the input layer followed by the LSTM layer.

```
decoder_input = Input(shape=(None, len(french_vocab)))
decoder_LSTM = LSTM(256, return_sequences=True, return_state = True)
```

- Get the output from the LSTM layer.

```
decoder_out, _ , _ = decoder_LSTM(decoder_input,
                                  initial_state=encoder_states)
```

- Feed LSTM output to a dense layer to get the final output.

```
decoder_dense = Dense(len(french_vocab), activation='softmax')
decoder_out = decoder_dense (decoder_out)
```

Combine the encoder and the decoder

- Combine encoder and decoder.

```
model = Model(inputs=[encoder_input, decoder_input], outputs=[decoder_out])
```

- Check model summary.

```
model.summary()
```

Compile and train the network

- Compile and train the model.

```
model.compile(optimizer='adam', loss='categorical_crossentropy')
model.fit(x=[input_data_prefix, input_data_suffix], y=target_data,
                  batch_size=64, epochs=1, validation_split=0.2)
```

Let's practice!

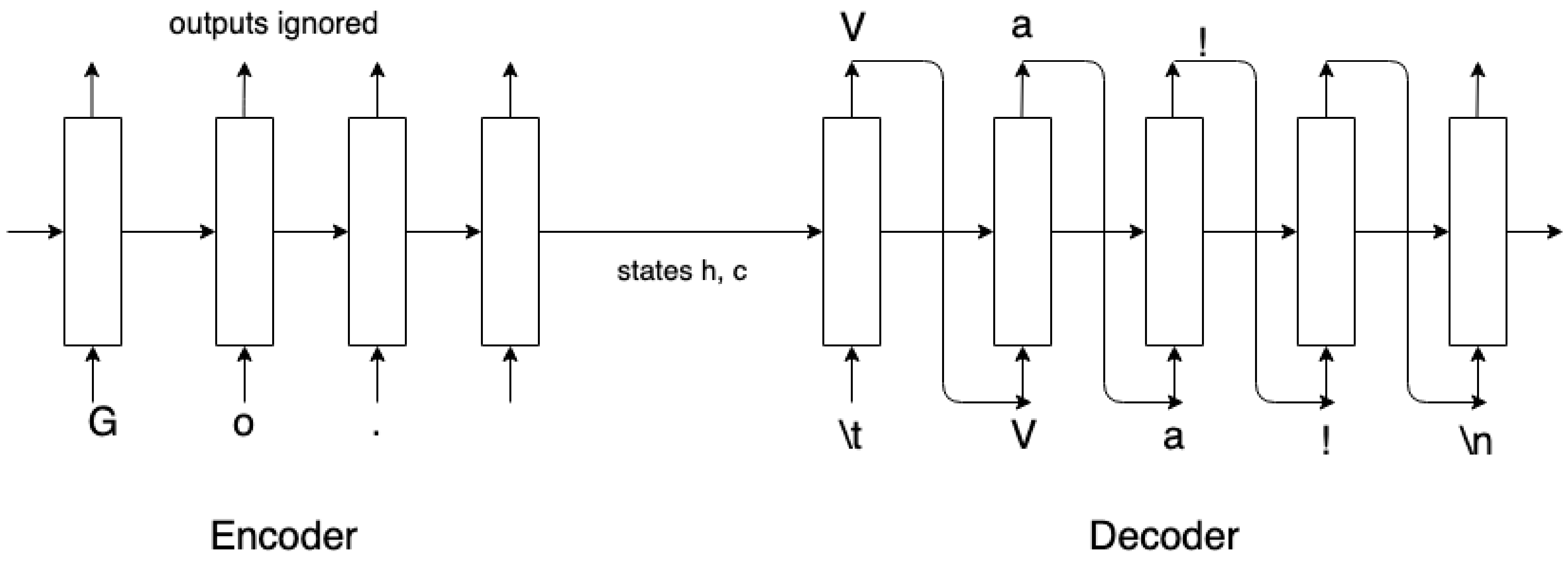
NATURAL LANGUAGE GENERATION IN PYTHON

Inference using encoder and decoder

NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

Encoder and decoder during inference



Inference model for encoder

- Encoder inference model.

```
encoder_model = Model(encoder_inputs, encoder_states)
```

Decoder initial states

- Define inputs of decoder inference model.

```
decoder_hidden_state = Input(shape=(latent_dim, None))
decoder_cell_state = Input(shape=(latent_dim, None))
```

- Initial state of the decoder LSTM layer.

```
decoder_input_states = [decoder_hidden_state, decoder_cell_state]
```

Decoder outputs

- Get decoder output from the trained decoder LSTM layer created earlier.

```
decoder_out, decoder_hidden, decoder_cell = decoder_LSTM(decoder_input,
                                                initial_state=decoder_input_states)
```

- Combine decoder states.

```
decoder_states = [decoder_hidden, decoder_cell]
```

- Feed decoder output to the trained decoder dense layer to predict the next character.

```
decoder_out = decoder_dense(decoder_out)
```

Inference model for decoder

- Decoder inference model.

```
# Define decoder inference model
decoder_model_inf = Model(inputs=[decoder_input] + decoder_input_states,
                          outputs=[decoder_out] + decoder_states )
```

Prediction using the inference models

- Pick an English sentence from the preprocessed English sentences.

```
inp_seq = tokenized_eng_sentences[10:11]
```

- Get encoder internal states

```
states_val = encoder_model.predict(inp_seq)
```

- Define variable to save output, initialized to contain the start token.

```
target_seq = np.zeros((1, 1, len(french_vocab)))
target_seq[0, 0, fra_char_to_index_dict['\t']] = 1
```

Generate the first character

- Get output from decoder inference model.

```
decoder_out, decoder_h, decoder_c = decoder_model_inf.predict(
                                    x=[target_seq] + states_val)
```

- Find index of most probable next character.

```
max_val_index = np.argmax(decoder_out[0,-1,:])
```

- Get actual character using index to character map.

```
sampled_suffix_char = idx_to_char[max_val_index]
```

Generate the second character

- Update target sequence and state values.

```
target_seq = np.zeros((1, 1, len(french_vocab)))
target_seq[0, 0, max_val_index] = 1
states_val = [decoder_h, decoder_c]
```

- Get output from decoder inference model.

```
decoder_out, decoder_h, decoder_c = decoder_model_inf.predict(
                              x=[target_seq] + states_val)
```

- Find most probable next character.

```
max_val_index = np.argmax(decoder_out[0,-1,:])
sampled_fra_char = fra_idx_to_char[max_val_index]
```

Generate translated sentence

```
translated_sent = ''
stop_condition = False
while not stop_condition:
    # Get decoder output
    decoder_out, decoder_h, decoder_c
                        = decoder_model.predict(x=[target_seq] + states_val)
    max_val_index = np.argmax(decoder_out[0,-1,:])
    # Append the generated character.
    translated_sent += fra_index_to_char_dict[max_val_index]
    # Stop if end token is encountered or max length reached
    if ( (sampled_fra_char == '\n') or (len(translated_sent) > max_len_fra_sent)):
        stop_condition = True
    # Store the generated character for next iteration
    target_seq = np.zeros((1, 1, len(french_vocab)))
    target_seq[0, 0, max_val_index] = 1
    states_val = [decoder_h, decoder_c]
print(translated_sent)
```

Let's practice!

NATURAL LANGUAGE GENERATION IN PYTHON