Convert email data to seq2seq

NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

Sentence auto-completion

- Input: an incomplete sentence.
- Output: a possible ending of the input sentence.
- Examples:
  - sentence: "Hi, How are you?"
  - input: "Hi, Ho", output: "w are you?"
  - input: "Hi, How ar", output: "e you?"

Email dataset

```
are we going to inspect tomorrow?
i will email you with the insurance info tomorrow.
steve, please remove bob shiring and liz rivera from rc 768. thank you phillip allen.
lucy,  the spreadsheet looks fine to me. phillip
please approve mike grigsby for bloomberg. thank you, phillip allen
i just refaxed. please confirm receipt
```

Prefixes and suffixes

- sentence: are we going to inspect tomorrow?

Prefix	Suffix
a	re we going to inspect tomorrow?
ar	e we going to inspect tomorrow?
are	we going to inspect tomorrow?
are w	e going to inspect tomorrow?
are we	going to inspect tomorrow?
. . .	. . .
are we going to inspect tomorro	w?
are we going to inspect tomorrow	?

Generate prefix and suffix sentences

```
prefix_sentences = []
suffix_sentences = []

# Iterate over each character position in each email
for email in corpus:
    for index in range(len(email)):
        # Find the prefix and suffix
        prefix = email[:index+1]
        suffix = '\t' + email[index+1:] + '\n'

        # Add prefix and sufix to the lists
        prefix_sentences.append(prefix)
        suffix_sentences.append(suffix)
```

Vocabulary and mappings

```
vocabulary = set(['\t', '\n'])

# Check each char in each email
for email in corpus:
    for char in email:
        # Add the char if not in vocabulary,
        if (char not in vocabulary):
            vocabulary.add(char)

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

# Create char to int and int to char mapping
char_to_idx = dict((char, idx) for idx, char in enumerate(vocabulary))
idx_to_char = dict((idx, char) for idx, char in enumerate(vocabulary))
```

Shape of input and target vectors

Define input and target vectors

```
# Find the length of the longest prefix and suffix
max_len_prefix_sent = max([len(prefix) for prefix in prefix_sentences])
max_len_suffix_sent = max([len(suffix) for suffix in suffix_sentences])
```

```
# Define a 3-D zero vector for the prefix sentences
input_data_prefix = np.zeros((len(prefix_sentences), max_len_prefix_sent,
                              len(vocabulary)), dtype='float32')
# Define a 3-D zero vector for the suffix sentences
input_data_suffix = np.zeros((len(suffix_sentences), max_len_suffix_sent,
                              len(vocabulary)), dtype='float32')
# Define a 3-D zero vector for the target data
target_data = np.zeros((len(suffix_sentences), max_len_suffix_sent,
                        len(vocabulary)), dtype='float32')
```

Initialize input and target vectors

```
for i in range(len(prefix_sentences)):
    # Iterate for each char in each prefix
    for k, ch in enumerate(prefix_sentences[i]):
        # Convert the char to one-hot encoded vector
        input_data_prefix[i, k, char_to_idx[ch]] = 1

    # Iterate for each char in each suffix
    for k, ch in enumerate(suffix_sentences[i]):
        # Convert the char to one-hot encoded vector
        input_data_suffix[i, k, char_to_idx[ch]] = 1

        # Target data is one timestep ahead and excludes start character
        if k > 0:
            target_data[i, k-1, char_to_idx[ch]] = 1
```

Let's practice!

NATURAL LANGUAGE GENERATION IN PYTHON

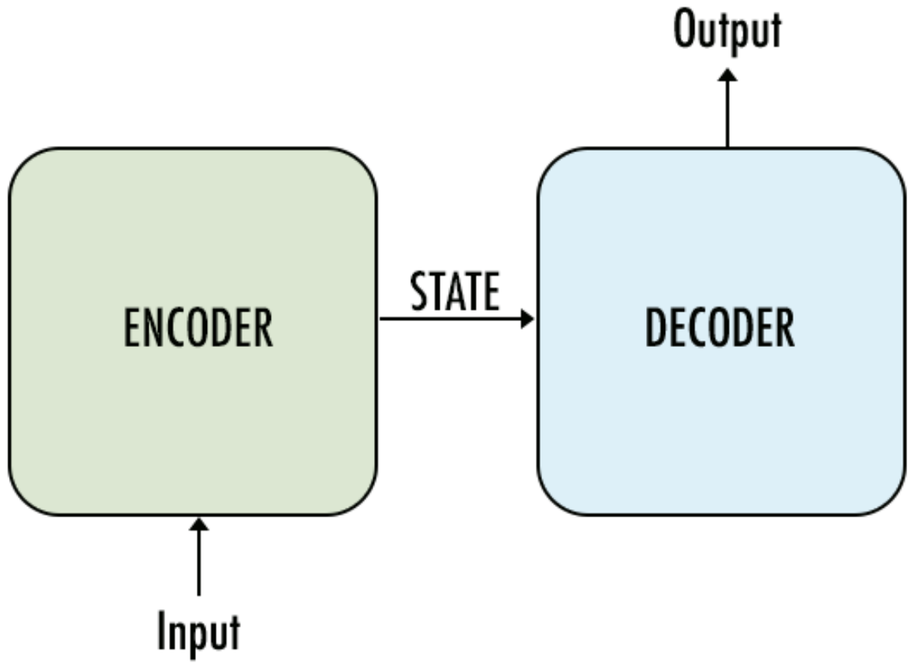
Sentence autocompletion using Encoder-Decoder

NATURAL LANGUAGE GENERATION IN PYTHON

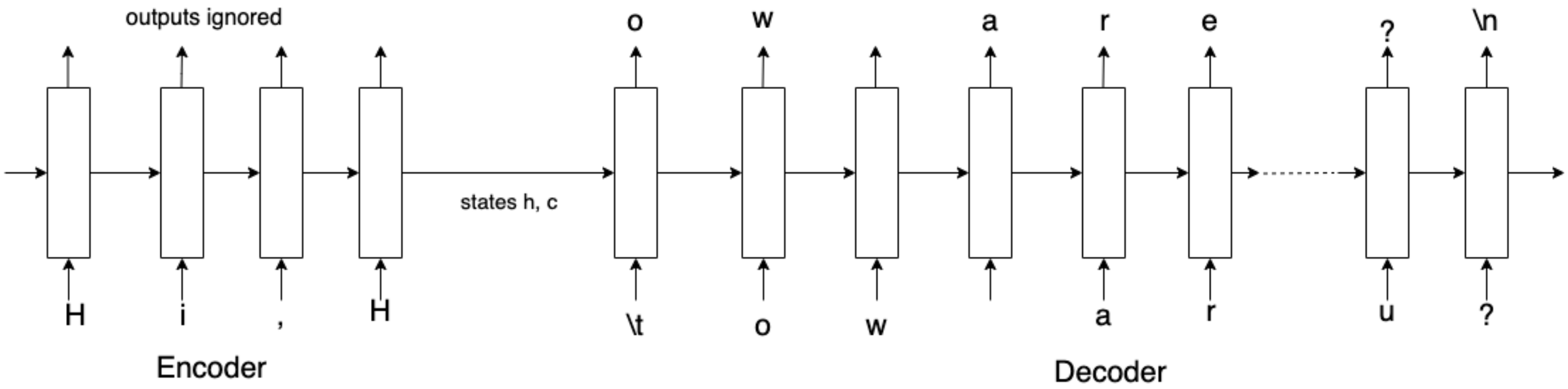
Biswanath Halder
Data Scientist

Encoder decoder architecture

- Encoder
  - Summarizes input information is states.
  - Outputs ignored.
- Decoder
  - Initial state - final state of the encoder.
  - Final states ignored.
  - Input during training - original target.
  - Input during inference - predicted target.



Encoder decoder for sentence auto-completion



Encoder for sentence auto-completion

```
# Create the input layer of the encoder
encoder_input = Input(shape=(None, len(vocabulary)))
```

```
# Create LSTM Layer of size 256
encoder_LSTM = LSTM(256, return_state = True)
```

```
# Save encoder output, hidden and cell state
encoder_outputs, encoder_h, encoder_c = encoder_LSTM(encoder_input)
```

```
# Save encoder states
encoder_states = [encoder_h, encoder_c]
```

Decoder for sentence auto-completion

```
decoder_input = Input(shape=(None, len(vocabulary)))
```

```
decoder_LSTM = LSTM(256, return_sequences=True, return_state = True)
```

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

```
decoder_dense = Dense(len(vocabulary), activation='softmax')
```

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

Combine the encoder and the decoder

- Build the model.

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

  - Check model summary.

```
model.summary()
```

Train the network

- Compile the model.

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

  - Train the model.

```
model.fit(x=[input_data_prefix, input_data_suffix], y=target_data,
                  batch_size=64, epochs=1, validation_split=0.2)
```

Let's practice!

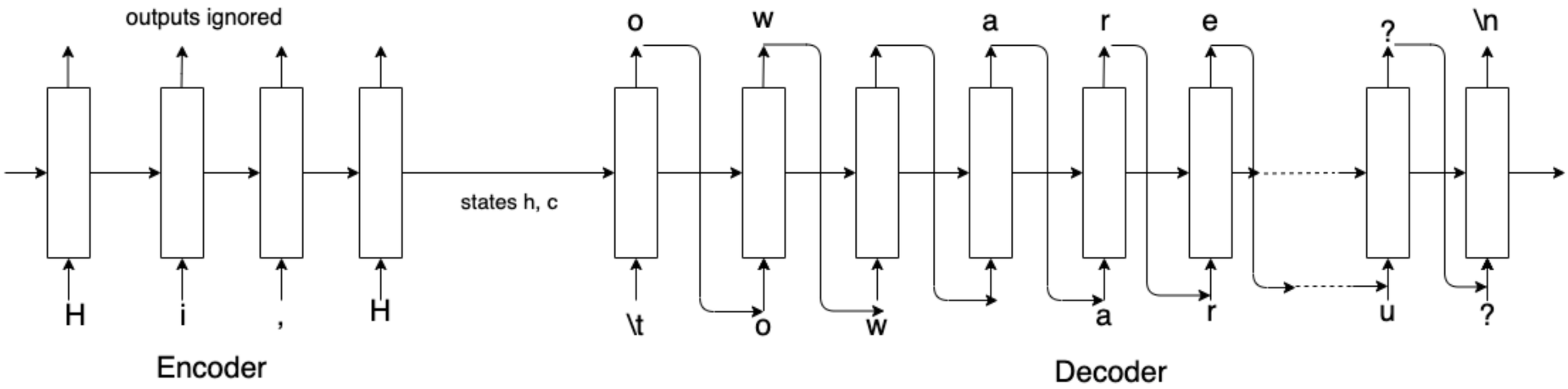
NATURAL LANGUAGE GENERATION IN PYTHON

Autocomplete sentences using inference models

NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

Encoder decoder during inference



Inference model for the encoder

- Encoder inference model.

```
encoder_model_inf = Model(encoder_input, encoder_states)
```

Initial states of decoder inference model

- Define hidden and cell states as inputs.

```
decoder_state_input_h = Input(shape=(256,))
decoder_state_input_c = Input(shape=(256,))
```

  - Concatenate the state vectors.

```
decoder_input_states = [decoder_state_input_h, decoder_state_input_c]
```

Output of decoder inference model

- Get output of decoder LSTM layer.

```
decoder_out, decoder_h, decoder_c = decoder_LSTM(decoder_input,
                                                 initial_state=decoder_input_states)
```

  - Save output states for next iteration.

```
decoder_states = [decoder_h , decoder_c]
```

  - Get probability distribution over vocabulary for next character.

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

Inference model for the decoder

- Decoder inference model.

```
decoder_model_inf = Model(inputs=[decoder_input] + decoder_input_states,
                          outputs=[decoder_out] + decoder_states )
```

Prediction using inference models

- Input a prefix into the encoder inference model.

```
inp_seq = input_data_prefix[4:5]
states_val = encoder_model_inf.predict(inp_seq)
```

  - Define variable for suffix.

```
target_seq = np.zeros((1, 1, len(vocabulary)))
```

  - Initialize the variable with the start token.

```
target_seq[0, 0, char_to_idx['\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 vectors.

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

  - Get decoder output.

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

  - Get most probable next character.

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

Auto-complete sentences

```
suffix_sent = ''
stop_condition = False
while not stop_condition:
    # Get output from decoder inference model
    decoder_out, decoder_h, decoder_c = decoder_model_inf.predict(x=[target_seq] + states_val)
    # Get next character and append it to the generated sequence
    max_val_index = np.argmax(decoder_out[0,-1,:])
    sampled_output_char = idx_to_char[max_val_index]
    suffix_sent += sampled_output_char
    # Check for end conditions
    if ((sampled_output_char == '\n') or (len(suffix_sent) > max_len_suffix_sent)) :
        stop_condition = True
    # Update target sequence and states values for next iteration
    target_seq = np.zeros((1, 1, len(vocabulary)))
    target_seq[0, 0, max_val_index] = 1
    states_val = [decoder_h, decoder_c]
```

Let's practice!

NATURAL LANGUAGE GENERATION IN PYTHON

Congratulations

NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

Chapter 1: Generate language using RNNs

- Recurrent neural networks.
- Baby name generation.

Chapter 2: Generate language using LSTMs

- Long short term memory.
- Generate text like Shakespeare.

Chapter 3: Machine translation

- Encoder-decoder architecture.
- Translation from English to French.

Chapter 4: Sentence auto-completion

- Sentence auto-completion.

Further studies

- Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network
- A Critical Review of Recurrent Neural Networks for Sequence Learning
- On the difficulty of training recurrent neural networks
- Learning to Forget: Continual Prediction with LSTM
- Sequence to Sequence Learning with Neural Networks

Advanced concepts

- Attention Is All You Need
- Effective Approaches to Attention-based Neural Machine Translation
- Transformer: A Novel Neural Network Architecture for Language Understanding

Thank you

NATURAL LANGUAGE GENERATION IN PYTHON