LSTM Sequence to Sequence Translation

Prof. Gheith Abandah

References:

- *Lstm seq2seq – Keras Documentation,* https://keras.io/examples/lstm_seq2seq/
- Code: https://github.com/keras-team/keras/blob/master/examples/lstm_seq2seq.py

Outline

1. Introduction
2. Summary of the algorithm
3. Imports and definitions
4. Vectorize the data
5. Create data structures
6. Build and train the model
7. Build the inference model
8. Decode sequences

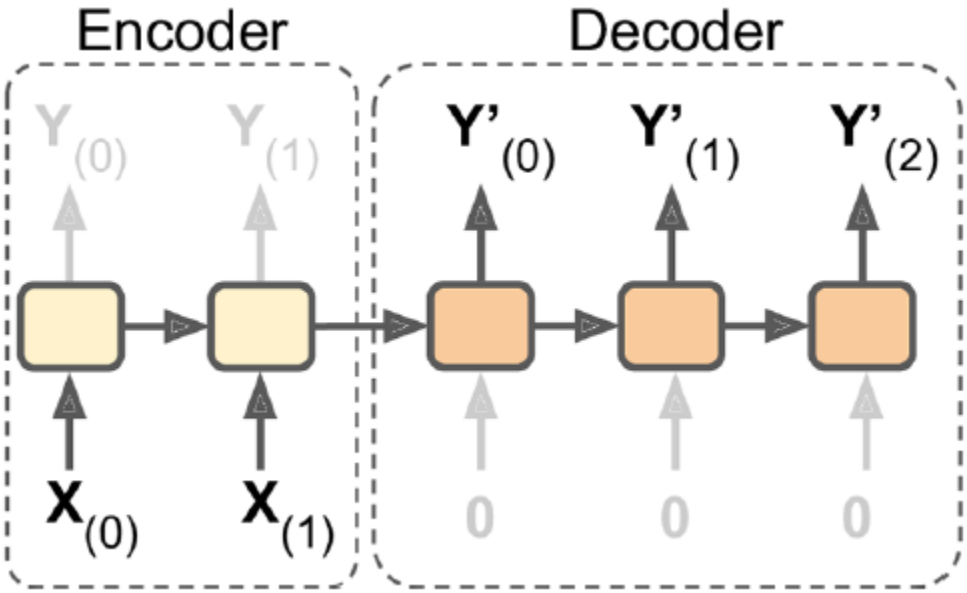
1. Introduction

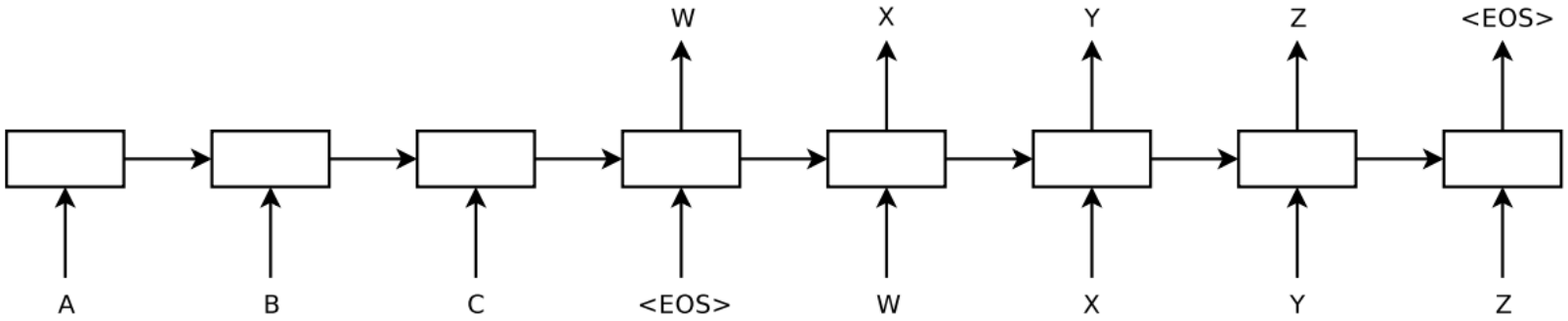
- Implements a basic character-level sequence-to-sequence model.
- Translate short English sentences into short Arabic sentences, character-by-character.
- The original example is English to French.
- Note that it is fairly unusual to do character-level machine translation, as word-level models are more common in this domain.

2. Summary of the Algorithm

- Start with input English sequences and corresponding target Arabic sequences.
- An encoder LSTM turns input sequences to 2 state vectors (we keep the last LSTM state and discard the outputs).
- A decoder LSTM is trained to turn the target sequences into the same sequence but offset by one time step in the future. It uses as initial state the state vectors from the encoder. Effectively, the decoder learns to generate `targets[t+1...]` given `targets[...t]`, conditioned on the input sequence.

2. Summary of the Algorithm





Ref: Sequence to Sequence Learning with Neural Networks, https://arxiv.org/pdf/1409.3215.pdf

2. Summary of the Algorithm

- In inference mode, when we want to decode unknown input sequences, we:
  - Encode the input sequence into state vectors
  - Start with a target sequence of size 1 (just the start-of-sequence character)
  - Feed the state vectors and 1-char target sequence to the decoder to produce predictions for the next character
  - Sample the next character using these predictions (we simply use argmax).
  - Append the sampled character to the target sequence
  - Repeat until we generate the end-of-sequence character or we hit the character limit.

Outline

1. Introduction
2. Summary of the algorithm
3. Imports and definitions
4. Vectorize the data
5. Create data structures
6. Build and train the model
7. Build the inference model
8. Decode sequences

3. Imports and Definitions

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

batch_size = 64       # Batch size for training.
epochs = 100          # Number of epochs to train for.
latent_dim = 256      # Number of cells.
num_samples = 20000   # Number of samples to train on.

# Path to the data txt file on disk.
data_path = 'ara-eng/ara.txt'
```

4. Vectorize the Data

```
input_texts = []
target_texts = []
input_characters = set()
target_characters = set()
with open(data_path, 'r', encoding='utf-8') as f:
    lines = f.read().split('\n')
for line in lines[: min(num_samples, len(lines) - 1)]:
    input_text, target_text = line.split('\t')
    target_text = '\t' + target_text + '\n'
    input_texts.append(input_text)
    target_texts.append(target_text)
    for char in input_text:
        if char not in input_characters:
            input_characters.add(char)
    for char in target_text:
        if char not in target_characters:
            target_characters.add(char)
```

```
Hi.     مرحبًا.
Run!    اركض!
Help!   النجدة!
Jump!   اقفز!
Stop!   قف!
```

4. Vectorize the Data

```
input_characters = sorted(list(input_characters))
target_characters = sorted(list(target_characters))
num_encoder_tokens = len(input_characters)
num_decoder_tokens = len(target_characters)
max_encoder_seq_length = max([len(txt) for txt in
        input_texts])
max_decoder_seq_length = max([len(txt) for txt in
        target_texts])


print('Number of samples:', len(input_texts))
print('Number of unique input tokens:', num_encoder_tokens)
print('Number of unique output tokens:',
        num_decoder_tokens)
print('Max sequence length for inputs:',
        max_encoder_seq_length)
print('Max sequence length for outputs:',
        max_decoder_seq_length)
```

4. Vectorize the Data

```
Number of samples: 11217
Number of unique input tokens: 73
Number of unique output tokens: 114
Max sequence length for inputs: 202
Max sequence length for outputs: 219
```

5. Create Data Structures

```
input_token_index = dict(
    [(char, i) for i, char in enumerate(input_characters)])
target_token_index = dict(
    [(char, i) for i, char in enumerate(target_characters)])


encoder_input_data = np.zeros( (len(input_texts),
      max_encoder_seq_length, num_encoder_tokens),
      dtype='float32')
decoder_input_data = np.zeros( (len(input_texts),
      max_decoder_seq_length, num_decoder_tokens),
      dtype='float32')
decoder_target_data = np.zeros( (len(input_texts),
      max_decoder_seq_length, num_decoder_tokens),
      dtype='float32')
```

5. Create Data Structures

```
for i, (input_text, target_text) in enumerate(zip(input_texts,
        target_texts)):
    for t, char in enumerate(input_text):
        encoder_input_data[i, t, input_token_index[char]] = 1.
    for t, char in enumerate(target_text):
        # target data is ahead of decoder input by 1 timestep
        decoder_input_data[i, t, target_token_index[char]] = 1.
        if t > 0:
            # decoder_target_data will be ahead by one timestep
             # and will not include the start character.
            decoder_target_data[i, t - 1,
              target_token_index[char]] = 1.
```

Outline

1. Introduction
2. Summary of the algorithm
3. Imports and definitions
4. Vectorize the data
5. Create data structures
6. Build and train the model
7. Build the inference model
8. Decode sequences

6. Build and Train the Model

```
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
encoder_states = [state_h, state_c]


decoder_inputs = Input(shape=(None, num_decoder_tokens))
decoder_lstm = LSTM(latent_dim, return_sequences=True,
       return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
       initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens,
       activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

6. Build and Train the Model

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


model.compile(optimizer='rmsprop',
      loss='categorical_crossentropy')
model.fit([encoder_input_data,
      decoder_input_data],
      decoder_target_data,
      batch_size=batch_size,
      epochs=epochs, validation_split=0.2)
```

6. Build and Train the Model

```
Train on 8973 samples, validate on 2244 samples
Epoch 1/100


  64/8973 [..............................] - ETA: 2:54
       - loss: 0.4223 - acc: 0.5851

…
8960/8973 [==============================>.] - ETA: 0s -
      loss: 0.1480 - acc: 0.0504
8973/8973 [==============================] - 80s
      9ms/step - loss: 0.1480 - acc: 0.0504 - val_loss:
      0.3615 - val_acc: 0.0705
```

Outline

1. Introduction
2. Summary of the algorithm
3. Imports and definitions
4. Vectorize the data
5. Create data structures
6. Build and train the model
7. Build the inference model
8. Decode sequences

7. Build the Inference Model

```
encoder_model = Model(encoder_inputs, encoder_states)


decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h,
      decoder_state_input_c]
decoder_outputs, state_h, state_c = decoder_lstm(
      decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)


decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs] + decoder_states)
```

7. Build the Inference Model

```
# Reverse-lookup token index to decode sequences
# back to something readable.
reverse_input_char_index = dict(
    (i, char) for char, i in
      input_token_index.items())
reverse_target_char_index = dict(
    (i, char) for char, i in
      target_token_index.items())
```

8. Decode Sequences

```
def decode_sequence(input_seq):
    states_value = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1, num_decoder_tokens))
    # Populate the first character of target sequence
       with the start character.
    target_seq[0, 0, target_token_index['\t']] = 1.

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
```

8. Decode Sequences

```
while not stop_condition:
        output_tokens, h, c = decoder_model.predict(
            [target_seq] + states_value)
        sampled_token_index = np.argmax(
            output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[
            sampled_token_index]
        decoded_sentence += sampled_char
        if (sampled_char == '\n' or
            len(decoded_sentence) >
            max_decoder_seq_length):
            stop_condition = True
        target_seq = np.zeros((1, 1,
            num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.
        states_value = [h, c]
    return decoded_sentence
```

8. Decode Sequences

```
for seq_index in range(100):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index
       + 1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Decoded sentence:', decoded_sentence)
```

8. Decode Sequences

```
-
Input sentence: Hi.
Decoded sentence: إنه يحب المانية التارية الماضية
      التارية الماضية.
-
Input sentence: Run!
Decoded sentence: أيمكنك أن تقول بالدرس الماضية.
```

Hi.	مرحبًا.
Run!	اركض!
Help!	النجدة!
Jump!	اقفز!
Stop!	قف!

Summary

1. Introduction
2. Summary of the algorithm
3. Imports and definitions
4. Vectorize the data
5. Create data structures
6. Build and train the model
7. Build the inference model
8. Decode sequences