Seq2seq with Attention

20191111382 소프트웨어융합학과 김세희

2023.12.17

1. 프로젝트 개요

✓ 프로젝트 목표
→ Seq2seq의 성능 향상 및 Attention과의 성능비교

✓ 프로젝트 기획 배경

→ 강의를 통해 Attention이 얼마나 강력한지 알 수 있었으나 직접적으로 비교하지는 않았다.

→ 프로젝트를 통해 Seq2seq에 Attention을 적용하여 성능을 비교해본다.

→ Seq2seq의 성능을 향상 시킬 수 있는 다른 기법을 찾아보고 실제로 효과가 있는지 성능을 확인해 본다.

2019111382 소프트웨어융합학과 김세희

1. 프로젝트 개요

<비교 분석을 수행할 모델 종류>

- 일반 데이터를 학습한 Seq2seq
- 반전된 데이터를 학습한 Seq2seq
- 일반 데이터를 학습한 Seq2seq with Attention
- 반전된 데이터를 학습한 Seq2seq with Attention

총 5가지 종류의 모델을 만들어 성능을 비교할 예정이다.

2. 데이터 설명

이전 Seq2seq 실습에서 사용한 영어-프랑스어 데이터셋을 사용

```
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
}

def download_zip(url, output_path):
    response = requests.get(url, headers=headers, stream=True)
    if response.status_code == 200:
        with open(output_path, 'wb') as f:
            for chunk in response.iter_content(chunk_size=8192):
                f.write(chunk)
        print(f"ZIP file downloaded to {output_path}")
    else:
        print(f"Failed to download. HTTP Response Code: {response.status_code}")

url = "http://www.manythings.org/anki/fra-eng.zip"
output_path = "fra-eng.zip"
download_zip(url, output_path)

path = os.getcwd()
zipfilename = os.path.join(path, output_path)

with zipfile.ZipFile(zipfilename, 'r') as zip_ref:
    zip_ref.extractall(path)


# 동일하게 20000개 사용
num_samples = 20000
def to_ascii(s):
  # 프랑스어 악센트(accent) 삭제
  # 예시 : 'déjà diné' -> deja dine
  return ''.join(c for c in unicodedata.normalize('NFD', s)
                 if unicodedata.category(c) != 'Mn')
```

2. 데이터 설명

전처리

```
def preprocess_sentence(sent):
  # 악센트 제거 함수 호출
  sent = to_ascii(sent.lower())

  # 단어와 구두점 사이에 공백 추가.
  # ex) "I am a student." => "I am a student ."
  sent = re.sub(r"([?.!,¿])", r" #1", sent)

  # (a-z, A-Z, ".", "?", "!", ",") 이들을 제외하고는 전부 공백으로 변환
  sent = re.sub(r"[^a-zA-Z!.?]+", r" ", sent)

  # 다수 개의 공백을 하나의 공백으로 치환
  sent = re.sub(r"#s+", " ", sent)
  return sent
```

```
def load_preprocessed_data():
  encoder_input, decoder_input, decoder_target = [], [], []

  with open("fra.txt", "r") as lines:
    for i, line in enumerate(lines):
      # source 데이터와 target 데이터 분리
      src_line, tar_line, _ = line.strip().split('#t')

      # source 데이터 전처리
      src_line = [w for w in preprocess_sentence(src_line).split()]

      # target 데이터 전처리
      tar_line = preprocess_sentence(tar_line)
      tar_line_in = [w for w in ("<sos> " + tar_line).split()]
      tar_line_out = [w for w in (tar_line + " <eos>").split()]

      encoder_input.append(src_line)
      decoder_input.append(tar_line_in)
      decoder_target.append(tar_line_out)

      if i == num_samples - 1:
        break

  return encoder_input, decoder_input, decoder_target

sents_en_in, sents_fra_in, sents_fra_out = load_preprocessed_data()
```

2. 데이터 설명

토큰화 및 패딩, 데이터 split(총 20,000개의 데이터를 사용)

```
# 토큰화 및 패딩 진행
tokenizer_en = Tokenizer(filters="", lower=False)
tokenizer_en.fit_on_texts(sents_en_in)
encoder_input = tokenizer_en.texts_to_sequences(sents_en_in)
encoder_input = pad_sequences(encoder_input, padding="post")

tokenizer_fra = Tokenizer(filters="", lower=False)
tokenizer_fra.fit_on_texts(sents_fra_in)
tokenizer_fra.fit_on_texts(sents_fra_out)

decoder_input = tokenizer_fra.texts_to_sequences(sents_fra_in)
decoder_input = pad_sequences(decoder_input, padding="post")

decoder_target = tokenizer_fra.texts_to_sequences(sents_fra_out)
decoder_target = pad_sequences(decoder_target, padding="post")

src_vocab_size = len(tokenizer_en.word_index) + 1
tar_vocab_size = len(tokenizer_fra.word_index) + 1
print("영어 단어 집합의 크기 : {:d}, 프랑스어 단어 집합의 크기 : {:d}".format(src_vocab_size, tar_vocab_size))
```

```
영어 단어 집합의 크기 : 3265, 프랑스어 단어 집합의 크기 : 5913
```

2. 데이터 설명

토큰화 및 패딩, 데이터 split(총 20,000개의 데이터를 사용)

```
# 단어로부터 정수를 얻는 딕셔너리와 정수로부터 단어를 얻는 딕셔너리를 각각 만듦
src_to_index = tokenizer_en.word_index
index_to_src = tokenizer_en.index_word
tar_to_index = tokenizer_fra.word_index
index_to_tar = tokenizer_fra.index_word

indices = np.arange(encoder_input.shape[0])
np.random.shuffle(indices)

encoder_input = encoder_input[indices]
decoder_input = decoder_input[indices]
decoder_target = decoder_target[indices]

n_of_val = int(20000*0.1)
encoder_input_train2 = encoder_input[:-n_of_val]
decoder_input_train2 = decoder_input[:-n_of_val]
decoder_target_train2 = decoder_target[:-n_of_val]

encoder_input_test2 = encoder_input[-n_of_val:]
decoder_input_test2 = decoder_input[-n_of_val:]
decoder_target_test2 = decoder_target[-n_of_val:]
```

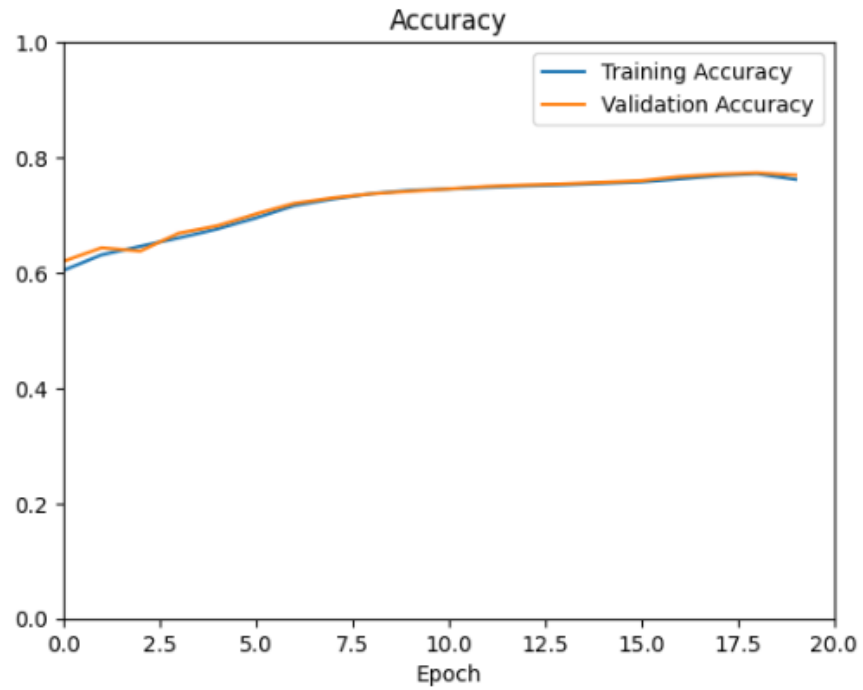
3. 모델 구축 – 기본 Seq2seq

```
# 인코더
encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(src_vocab_size, embedding_dim)(encoder_inputs) # 임베딩 층
enc_masking = Masking(mask_value=0.0)(enc_emb) # 패딩 0은 연산에서 제외

# 상태값 리턴을 위해 return_state는 True
encoder_lstm = LSTM(hidden_units, activation='relu', dropout=0.2, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(enc_masking) # 은닉 상태와 셀 상태를 리턴
encoder_states = [state_h, state_c] # 인코더의 은닉 상태와 셀 상태를 저장

# 디코더
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(tar_vocab_size, hidden_units) # 임베딩 층
dec_emb = dec_emb_layer(decoder_inputs)
dec_masking = Masking(mask_value=0.0)(dec_emb) # 패딩 0은 연산에서 제외

# 상태값 리턴을 위해 return_state는 True,
# 모든 시점에 대해서 단어를 예측하기 위해 return_sequences는 True
decoder_lstm = LSTM(hidden_units,
                    activation='relu',
                    dropout=0.2,
                    return_sequences=True,
                    return_state=True)

# 인코더의 은닉 상태를 초기 은닉 상태(initial_state)로 사용
decoder_outputs, _, _ = decoder_lstm(dec_masking,
                                     initial_state=encoder_states)

# 모든 시점의 결과에 대해서 소프트맥스 함수를 사용한 출력층을 통해 단어 예측
decoder_dense = Dense(tar_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

- embedding_dim = 64
- hidden_units = 64

3. 모델 구축 – 기본 Seq2seq

```
# 동일한 모델에 데이터를 바꿔서 실행해보자!
model3 = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])

hist3 = model3.fit(x=[encoder_input_train2, decoder_input_train2], y=decoder_target_train2, #
          validation_data=([encoder_input_test2, decoder_input_test2], decoder_target_test2), batch_size=128, epochs=20)
```



- batch_size = 128
- epoch = 20
- learning_rate = 0.001
- 최종 정확도: 약 76%

3. 모델 구축 – 반전된 데이터를 학습한 Seq2seq

✓ 기획 배경

- Sutskever et al. "Sequence to sequence learning with neural networks."(2014)
- 요약: 입력 데이터를 거꾸로 반전시키면 LSTM의 성능이 향상한다.
- "단어의 순서를 거꾸로 하면서 대부분 학습 진행이 빨라지더라"

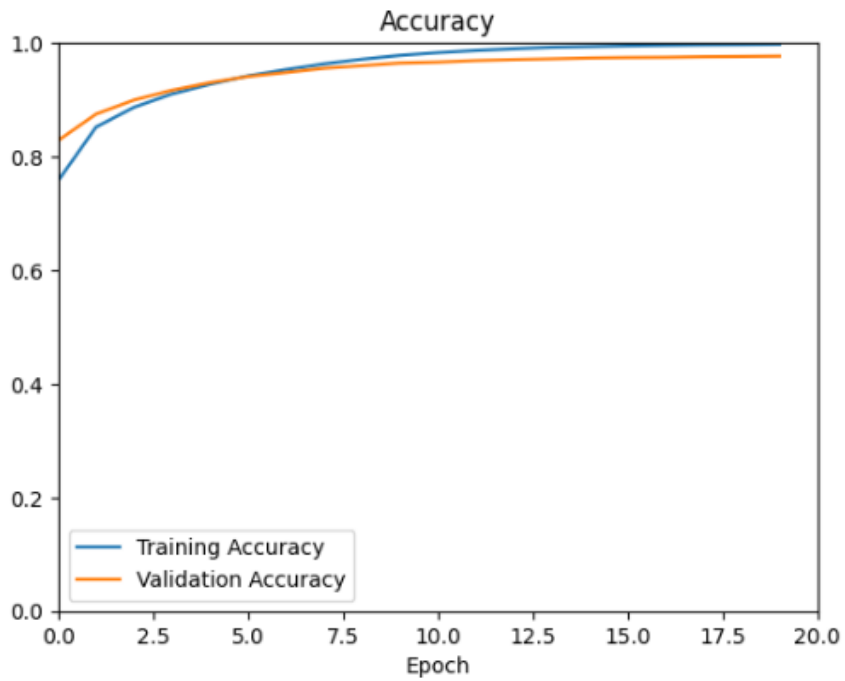
```
# 동일하게 입력 데이터 반전도 수행해보자
re_encoder_input_train2 = encoder_input_train2[:, ::-1]
re_decoder_input_train2 = decoder_input_train2[:, ::-1]
re_decoder_target_train2 = decoder_target_train2[:, ::-1]

re_encoder_input_test2 = encoder_input_test2[:, ::-1]
re_decoder_input_test2 = decoder_input_test2[:, ::-1]
re_decoder_target_test2 = decoder_target_test2[:, ::-1]
```

```
# 체크
print(decoder_input_train[1])
print(re_decoder_input_train[1])
```

```
[   1   18  192  427  115   24  165 5590 1463 1040    7    3  133   15
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0]
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0   15  133    3    7 1040 1463 5590  165   24  115  427  192
   18    1]
```

3. 모델 구축 – 반전된 데이터를 학습한 Seq2seq

```
model4 = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model4.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])

hist4 = model4.fit(x=[re_encoder_input_train2, re_decoder_input_train2], y=re_decoder_target_train2, #
          validation_data=([re_encoder_input_test2, re_decoder_input_test2], re_decoder_target_test2), batch_size=128, epochs=20)
```



- batch_size = 128
- epoch = 20
- learning_rate = 0.001
- 최종 정확도: 약 99%

3. 모델 구축 – 일반 데이터를 학습한 Seq2seq with Attention

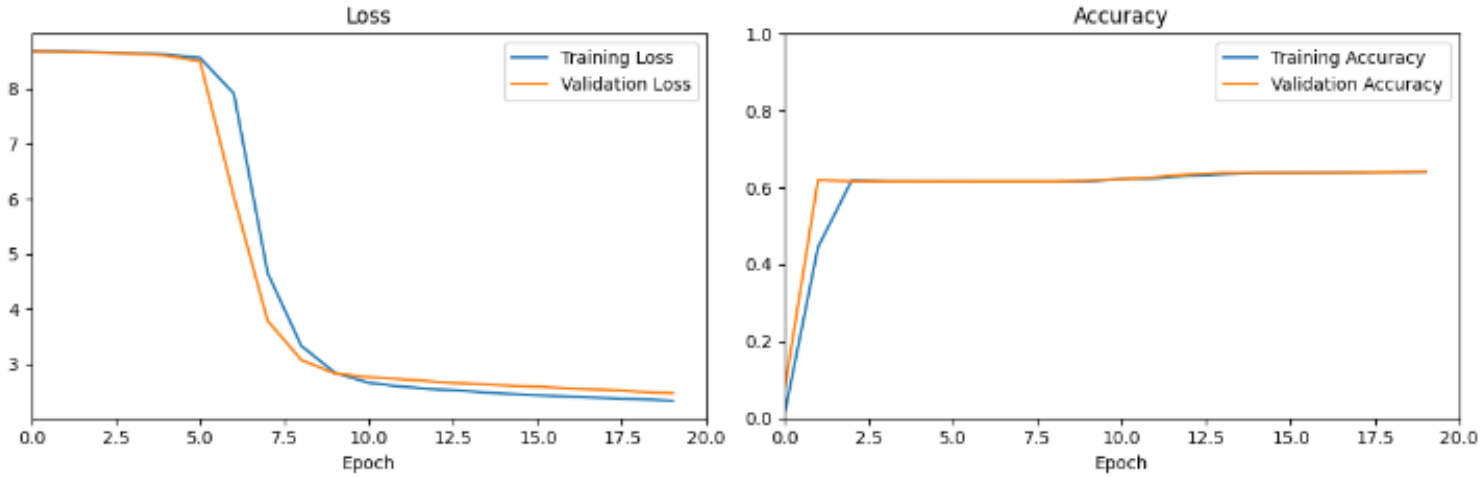
```
# 디코더
decoder_inputs = Input(shape=(None, ))
dec_emb_layer = Embedding(tar_vocab_size, embedding_dim)
dec_emb = dec_emb_layer(decoder_inputs)
dec_masking = Masking(mask_value=0.0)(dec_emb)

decoder_lstm = LSTM(hidden_units,
                    activation='relu',
                    dropout=0.2,
                    return_sequences=True,
                    return_state=True)

Query, h, c = decoder_lstm(dec_masking, initial_state=encoder_states)

# Attention 적용
attention_layer = Attention()

key_value = tf.concat([encoder_states[0][:, tf.newaxis, :], Query[:, :-1, :]], axis=1)

attention_output = attention_layer([key_value, encoder_outputs])
attention_output = tf.concat([Query, attention_output], axis=-1)

# 소프트맥스 함수를 사용한 출력층을 통해 단어 예측
decoder_dense = Dense(tar_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(attention_output)

# loss가 자꾸 발산해서 learning rate을 조금 낮춤 0.001 -> 0.00001
optimizer = keras.optimizers.Adam(learning_rate=0.00001)
```

<변경 사항>

- 디코더의 LSTM과 Dense 사이에 *Attention layer* 추가
- 학습할 때 loss가 nan으로 발산하려고 해서 이전에 일반적인 Seq2seq보다 learning rate을 낮춤
  - 0.001 -> 0.00001

3. 모델 구축 – 일반 데이터를 학습한 Seq2seq with Attention

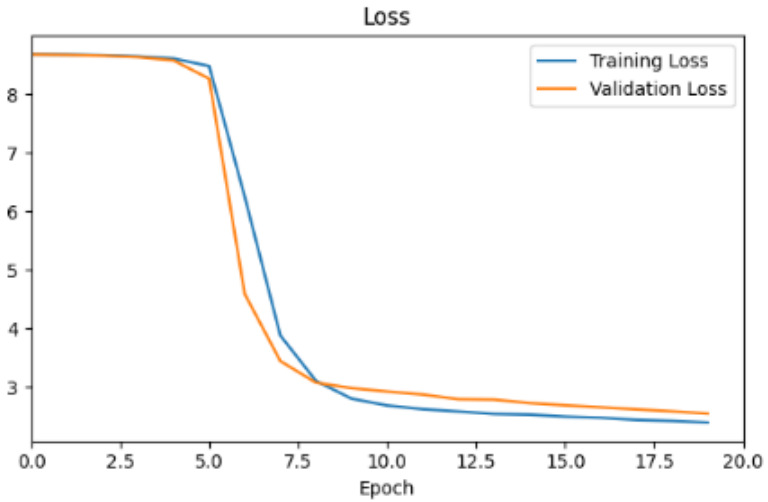
```
# 모델의 입력과 출력을 정의.
model_with_attention = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model_with_attention.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['acc'])
hist5 = model_with_attention.fit(x=[encoder_input_train, decoder_input_train], y=decoder_target_train, ₩
          validation_data=([encoder_input_test, decoder_input_test], decoder_target_test), batch_size=128, epochs=20)
```

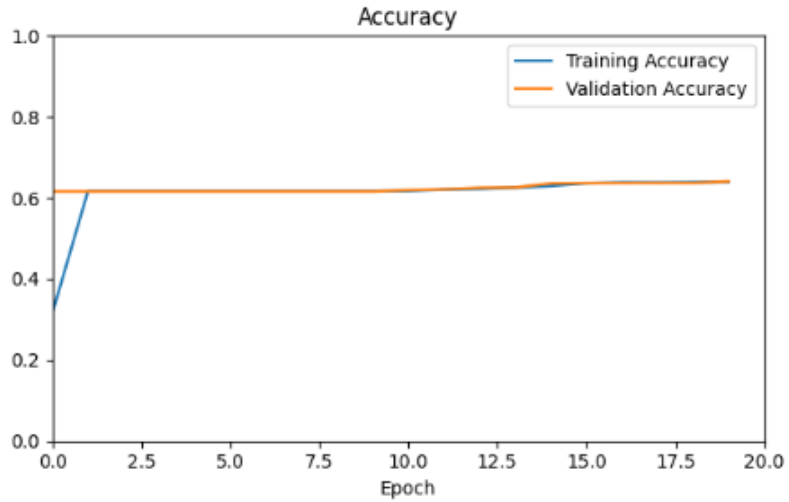


- batch_size = 128
- epoch = 20
- learning_rate = 0.00001
- 최종 정확도: 약 65%

3. 모델 구축 – 반전된 데이터를 학습한 Seq2seq with Attention

```
# 영어 - 프랑스어 뒤집은 데이터
model_with_attention4 = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model_with_attention4.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['acc'])
hist10 = model_with_attention4.fit(x=[re_encoder_input_train2, re_decoder_input_train2], y=re_decoder_target_train2, #
          validation_data=([re_encoder_input_test2, re_decoder_input_test2], re_decoder_target_test2), batch_size=128, epochs=20)
```

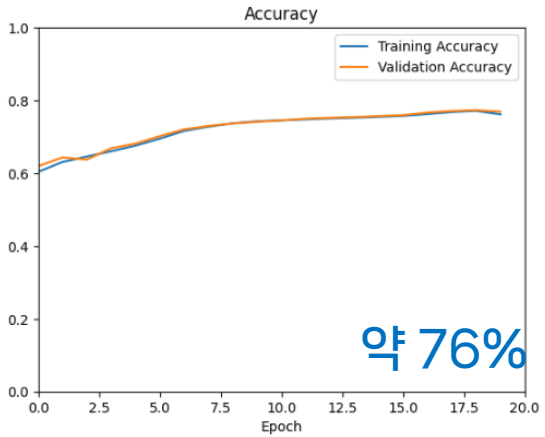


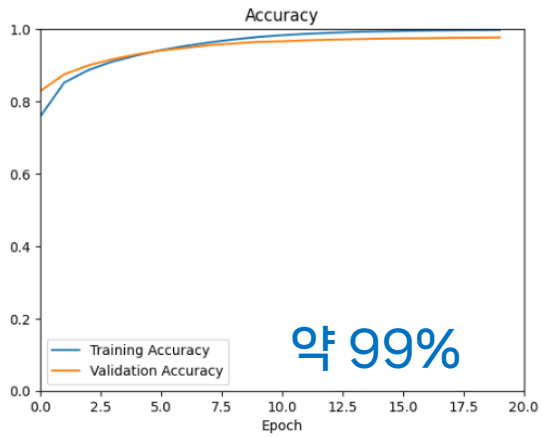


- batch_size = 128
- epoch = 20
- learning_rate = 0.00001
- 최종 정확도: 약 49%

4. 모델 비교 평가

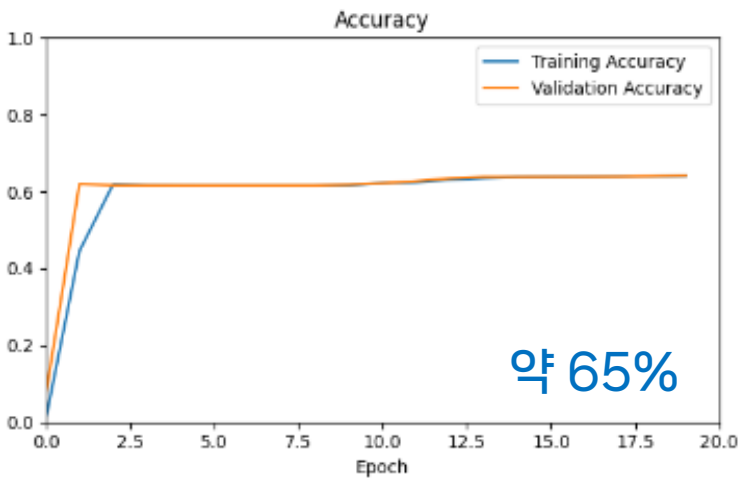
일반 데이터
Seq2seq

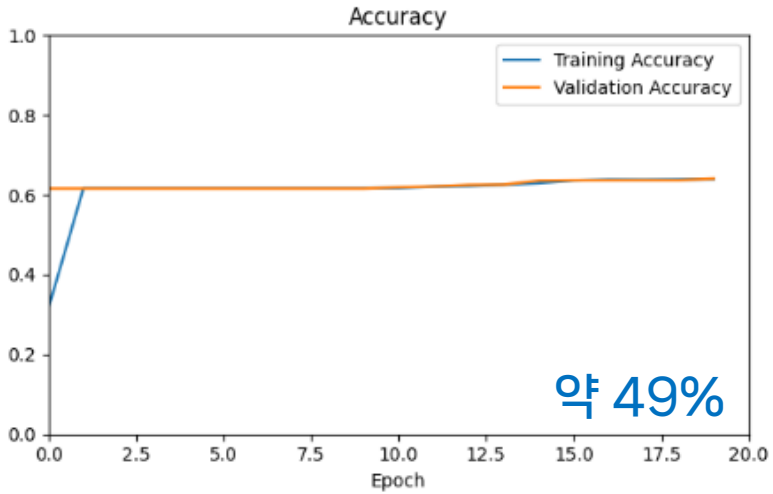




반전 데이터
Seq2seq

일반 데이터
Seq2seq with
attention





반전 데이터
Seq2seq with
attention

5. 결론

1. Attention을 적용하면 성능이 무조건 향상 될 것이라고 생각했으나 그러지 않았다.

<추정되는 원인>

- Attention 층을 구현할 때 잘못 연결했다.
- 데이터가 부족하다. (총 20,000개라고 하지만 split을 하면 실질적으로 18,000개)

2. Sutskever et al. "Sequence to sequence learning with neural networks."(2014)에서 제안한 데이터 반전은 충분한 효과를 보였다.

정확도가 일반 Seq2seq보다 빠르게 수렴하고 동일 epoch 학습 시 더 높은 정확도를 보여주었다.

번외. 한국어 데이터 사용

번외 1. 기존 프로젝트 개요 – 제주도 방언 번역기

한국 표준어와 다른 형태인 방언 중에서도

일반인들에게 가장 익숙하지 않은 제주도 방언 번역기를 제작하고자 했다.

번외 2. 제주도 방언 번역기의 데이터

✓ 이전 과정과 동일한 모델에 아닌 표준어-제주도 방언 데이터를 학습 시키기 위해 수집 및 정제가 필요했다.

✓ 데이터는 AI hub의 한국어 방언 발화 데이터를 다운로드 받아서 Json parsing 이후에 사용했다.


```
{
    "id": "DZES20000002.1.1.7",
    "start": 9.86,
    "end": 10.83,
    "speaker_id": "2",
    "form": "예전에 (경)/(그렇게)",
    "standard_form": "예전에 그렇게",
    "dialect_form": "예전에 경",
    "note": "",
    "eojeolList": [
        {
            "id": 1,
            "eojeol": "예전에",
            "standard": "예전에",
            "isDialect": false
        },
        {
            "id": 2,
            "eojeol": "경",
            "standard": "그렇게",
            "isDialect": true
        }
    ]
},
{
    "id": "DZES20000002.1.1.8",
    "start": 10.84,
    "end": 12.95,
    "speaker_id": "2",
    "form": "해놨잖아 작은오빠",
    "standard_form": "해놨잖아 작은오빠",
    "dialect_form": "해놨잖아 작은오빠",
    "note": "",
    "eojeolList": [
        {
            "id": 1,
            "eojeol": "해놨잖아",
            "standard": "해놨잖아",
            "isDialect": false
```

번외 2. 제주도 방언 번역기의 데이터

- Json Parsing

```
folder_path = "/content/drive/MyDrive/Data/Data/딥러닝 및 응용_제주도 방언/"
```

```
def parse_dialect_data(json_data):
    parsed_data = json.loads(json_data)
    dialect_entries = []

    for utterance in parsed_data.get("utterance", []):
        eojeol_list = utterance.get("eojeolList", [])
        for eojeol in eojeol_list:
            if eojeol.get("isDialect", False):
                standard_form = utterance.get("standard_form", "")
                dialect_form = utterance.get("dialect_form", "")
                dialect_entries.append({"standard_form": standard_form, "dialect_form": dialect_form})

    return dialect_entries

def parse_json_files_in_folder(folder_path):
    parsed_entries = []

    for filename in os.listdir(folder_path):
        if filename.endswith(".json"):
            file_path = os.path.join(folder_path, filename)
            with open(file_path, "r", encoding="utf-8") as file:
                json_data = file.read()
                entries = parse_dialect_data(json_data)
                parsed_entries.extend(entries)

    return parsed_entries

parsed_entries = parse_json_files_in_folder(folder_path)
```

```
# 중복 제거하기
unique_entries = [dict(t) for t in {tuple(d.items()) for d in parsed_entries}]
```

번외 2. 제주도 방언 번역기의 데이터

- 데이터가 충분히 많았기 때문에(약 20만개) 불필요한 특수 문자가 있는 text는 모두 제거 했다.

```
# 특수문자가 없는 요소만 가져와서 list 만들기
def parse_and_clean_entries(entries):
    cleaned_entries = []

    for entry in entries:
        standard_form = entry['standard_form']
        dialect_form = entry['dialect_form']

        # 괄호와 특수문자를 포함하지 않는 경우에만 추가
        if not re.search(r'[₩(₩)₩[₩]₩{₩}₩<₩>₩!₩@₩₩₩$₩%₩^₩&₩*₩-₩+₩=₩_₩₩₩/₩?₩,₩'₩"₩.₩:₩;₩|₩`₩~₩@₩₩₩*₩&₩^₩-]', standard_form+dialect_form):
            cleaned_entries.append({"standard_form": standard_form, "dialect_form": dialect_form})

    return cleaned_entries
```

```
# 데이터 자체가 많기 때문에, 특수 문자가 있는 문장은 다 제거한다.
clean_text = parse_and_clean_entries(unique_entries)

# 표준어와 방언은 각각의 리스트로 분리한다.
standard_forms = [entry["standard_form"] for entry in clean_text]
dialect_forms = [entry["dialect_form"] for entry in clean_text]
```

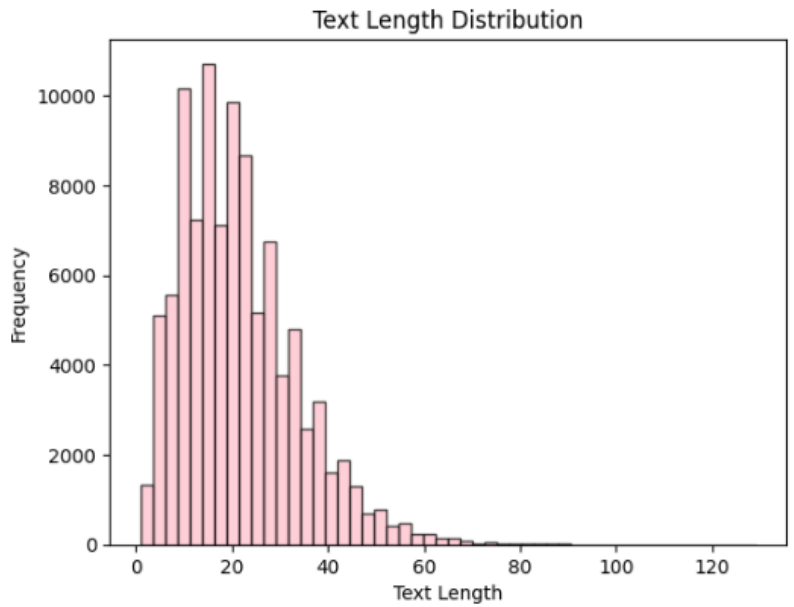
```
# 데이터의 길이 확인
print("표준어 데이터의 길이:", len(standard_forms))
print("방언 데이터의 길이:", len(dialect_forms))
```

```
표준어 데이터의 길이: 100296
방언 데이터의 길이: 100296
```

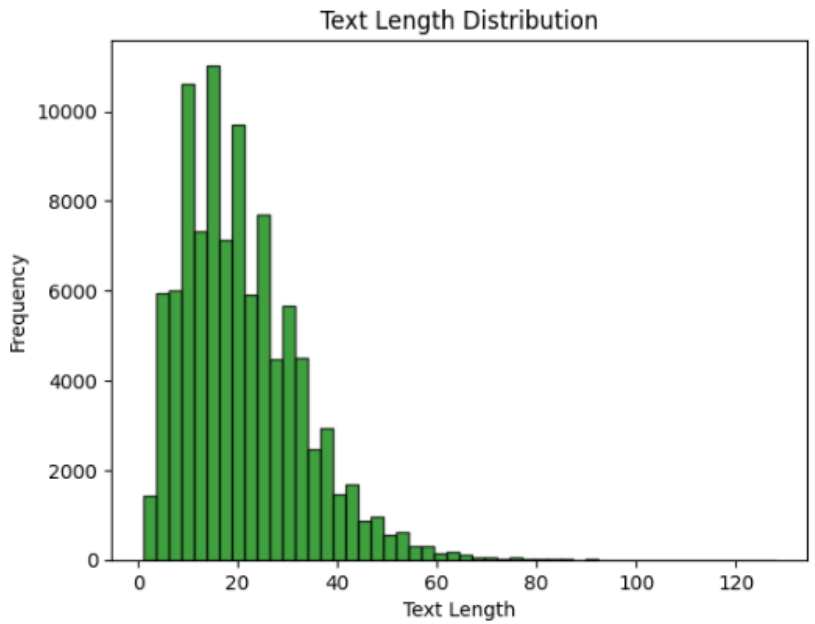
번외 2. 제주도 방언 번역기의 데이터

- 간단한 EDA

```
# EDA - 표준어 데이터의 문장 길이 분포
standard_text_lengths = [len(text) for text in standard_forms]
dialect_text_lengths = [len(text) for text in dialect_forms]

plt.hist(standard_text_lengths, bins=50, alpha=0.75, color='pink', edgecolor='black')
plt.title('Text Length Distribution')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.show()
```




```
# EDA - 방언 데이터의 문장 길이 분포
plt.hist(dialect_text_lengths, bins=50, alpha=0.75, color='g', edgecolor='black')
plt.title('Text Length Distribution')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.show()
```



번외 2. 제주도 방언 번역기의 데이터

```
# 토큰화 및 정수 인코딩
from konlpy.tag import Okt

tokenizer = Okt()

# 토큰화 함수 정의 -> okt를 통한 한국어 토크나이징
def tokenize(texts):
    return [tokenizer.morphs(text) for text in texts]

# 표준어와 방언 데이터를 X와 y로 나누기, 총 20000개 사용
X = standard_forms[:20000]
y = dialect_forms[:20000]

# 표준어와 방언 데이터를 토큰화
X_tokenized = tokenize(X)
y_tokenized = tokenize(y)

# SOS와 EOS 토큰 추가
sos_token = '<sos>'
eos_token = '<eos>'

# 디코더의 예측, 학습을 위한 데이터 각각 만들기
y_tokenized_input = [[sos_token] + seq for seq in y_tokenized] # 학습
y_tokenized_target = [seq + [eos_token] for seq in y_tokenized] # 예측

# 토큰을 정수로 변환하는 Tokenizer 생성
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(X_tokenized)
x_encoded = x_tokenizer.texts_to_sequences(X_tokenized)

y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(y_tokenized_input + y_tokenized_target)
y_encoded_input = y_tokenizer.texts_to_sequences(y_tokenized_input)
y_encoded_target = y_tokenizer.texts_to_sequences(y_tokenized_target)

# 패딩 추가, pre가 성능이 더 좋다는 말이 있지만 일단 post
encoder_input = pad_sequences(x_encoded, padding='post')
decoder_input = pad_sequences(y_encoded_input, padding='post')
decoder_target = pad_sequences(y_encoded_target, padding='post')
```

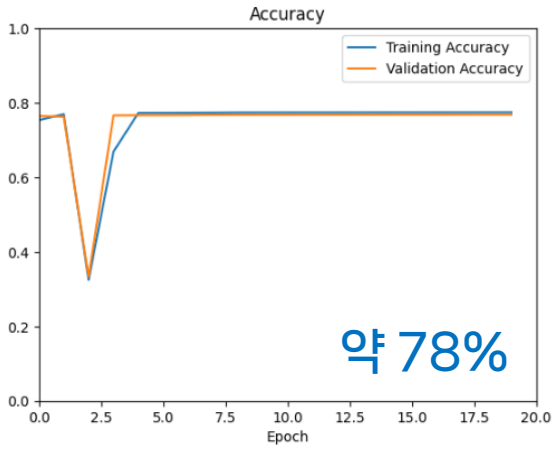
```
# 학습, 검증, 테스트 데이터로 나누기
n_of_val = int(20000*0.1)
encoder_input_train = encoder_input[:-n_of_val]
decoder_input_train = decoder_input[:-n_of_val]
decoder_target_train = decoder_target[:-n_of_val]

encoder_input_test = encoder_input[-n_of_val:]
decoder_input_test = decoder_input[-n_of_val:]
decoder_target_test = decoder_target[-n_of_val:]

# 각 데이터셋의 크기 확인
print('훈련 source 데이터의 크기 :',encoder_input_train.shape)
print('훈련 target 데이터의 크기 :',decoder_input_train.shape)
print('훈련 target 레이블의 크기 :',decoder_target_train.shape)
print('테스트 source 데이터의 크기 :',encoder_input_test.shape)
print('테스트 target 데이터의 크기 :',decoder_input_test.shape)
print('테스트 target 레이블의 크기 :',decoder_target_test.shape)
```

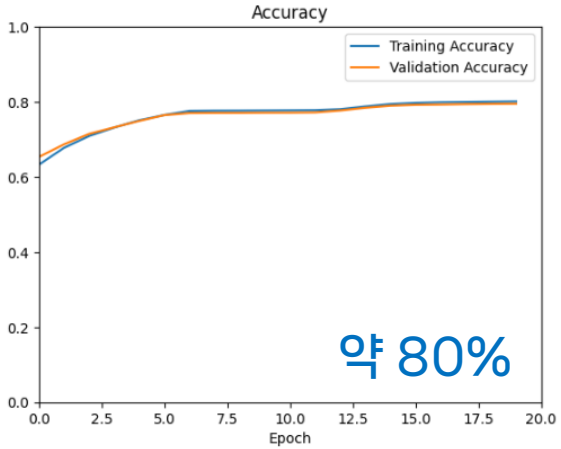
```
훈련 source 데이터의 크기 : (18000, 44)
훈련 target 데이터의 크기 : (18000, 44)
훈련 target 레이블의 크기 : (18000, 44)
테스트 source 데이터의 크기 : (2000, 44)
테스트 target 데이터의 크기 : (2000, 44)
테스트 target 레이블의 크기 : (2000, 44)
```

2019111382 소프트웨어융합학과 김세희

번외 3. 제주도 방언 번역기 모델 비교

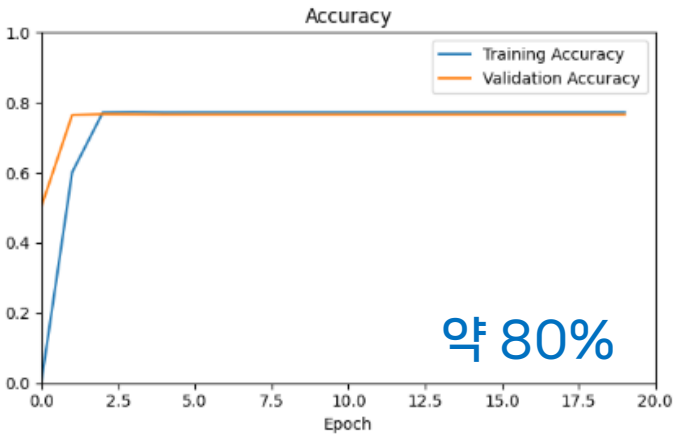
일반 데이터
Seq2seq

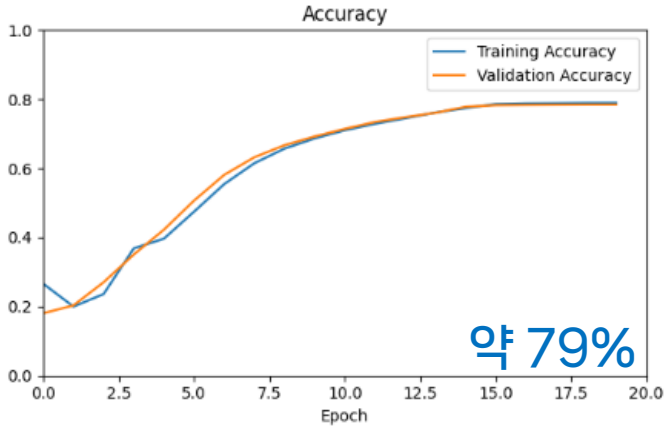




반전 데이터
Seq2seq

일반 데이터
Seq2seq with
attention





반전 데이터
Seq2seq with
attention

6. 프로젝트 종합 결론

1. 모델이 어느 일정 수준에 도달하면 정확도가 상승하지 않았다.

✓ 한국어는 tokenizing 방식에 따라 모델의 성능을 향상시킬 수 있다는 선행 연구가 존재한다.

✓ Park et al. "An empirical study of tokenization strategies for various Korean NLP tasks."(2020) *arXiv:2010.02534*.

→ 본 프로젝트에서는 tokenizer로 Okt를 사용했지만 다른 tokenize 방식을 사용한다면 성능이 향상될지도 모른다.

→ 성능을 향상시키기 위해 데이터 보완을 해본다.

2. 영어-프랑스어 데이터 셋에서는 Attention의 성능이 일반 seq2seq에 밀리는 모습을 보였지만 표준어-방언 데이터 셋에서는 정확도가 유사하고 수렴이 훨씬 안정적이다.

✓ 영어-프랑스어 데이터에서 성능이 안 좋았던 원인을 데이터로 좁혀볼 수 있다.

감사합니다.

2023.12.17