# Sequence to Sequence Learning with Neural Networks

Mulcam_b

김준수, 이규호, 이수범, 이예슬, 염성현

# INDEX

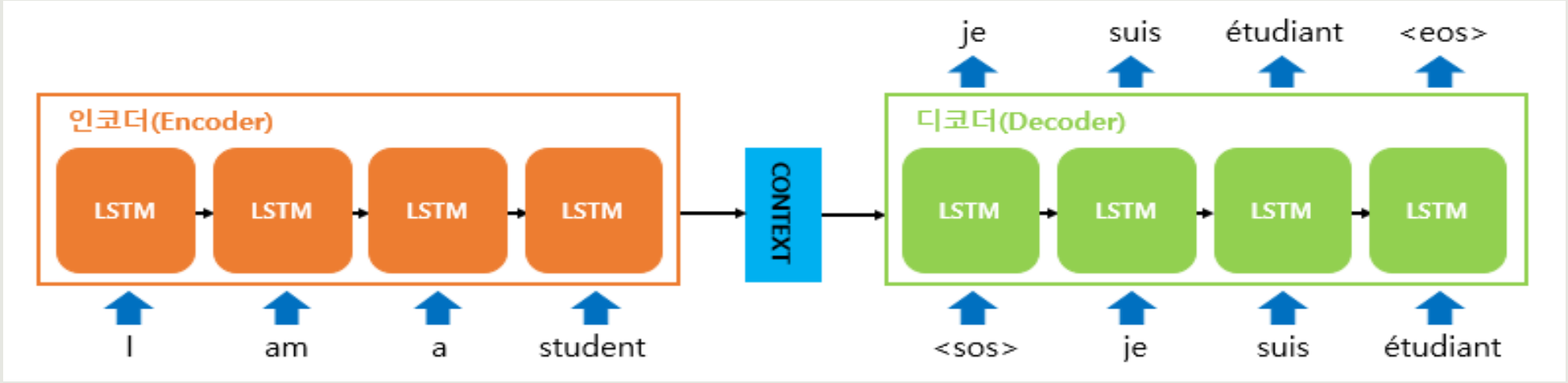
*Sequence to Sequence Learning with Neural Networks*

# 논문

*Sequence to Sequence Learning with Neural Networks*

## LSTM

(seq2seq)




## 특징

**Input Sequence, Output Sequence 2개의 다른 LSTM 사용**

**4개의 Layer사용**

**학습과정에서 입력문장의 순서를 변경**

# 논문

*Sequence to Sequence Learning with Neural Networks*

## Training Data

1. 파라미터는 (-0.08, 0.08) 사이의 **uniform distribution(균일분포)** 이루도록 정규화

2. Optimizer는 **SGD** 사용

3. Learning rate = 0.7, 5 epochs 후에는 0.5 epochs 마다 절반으로 줄임

-> **총 7.5 epochs으로 학습**

4. **128 Sequence** 크기의 배치 사용

5. 평균 gradient의 L2 norm값이 5를 넘어갈 경우,
   Clipping Gradient를 사용해서 5 이하로 조절

6. 미니 배치를 나눌 때 비슷한 길이로 추출

   ▶ 패딩으로 인한 공간낭비를 줄임

# PyTorch 모델 구현 - Tokenize

*Sequence to Sequence Learning with Neural Networks*

## Tokenize

영문은 Spacy, 한국어는 Mecab을 이용

## 토큰화 시 주의점

1. 구두점 & 특수 문자
   ▶ Ph.D, AT&T

2. 줄임말, 단어 내 띄어쓰기
   ▶ New York, ASAP

```
41
42 # 테스트
43 tokenize_en("I like new york in autumn and my company is Alphabet Inc.")

['I', 'like', 'new york', 'in', 'autumn', 'and', 'my', 'company', 'is', 'Alphabet Inc.']
```

```
def tokenize_en(text):#영어 단어 토큰화 후 리스트로
    return [tok.text for tok in spacy_en.tokenizer(text)]

def tokenize_kor(text):# 한국어 단어 토큰화 후 리스트로
    return mecab_kor.morphs(text)
```

```
class MatchRetokenizeComponent:
  def __init__(self, nlp, terms):
    self.terms = terms
    self.matcher = PhraseMatcher(nlp.vocab)
    patterns = [nlp.make_doc(text) for text in terms]
    # Only run nlp.make_doc to speed things up
    self.matcher.add("TerminologyList", None, *patterns)
    Doc.set_extension("phrase_matches", getter=self.matcher, force=True) # You shou

  def __call__(self, doc):
    matches = self.matcher(doc)
    with doc.retokenize() as retokenizer:
        for match_id, start, end in matches:
            retokenizer.merge(doc[start:end], attrs={"LEMMA": str(doc[start:end])})
    return doc

# 영어 토큰화
def tokenize_en(text):
  # extract entity
  nlp_en = spacy.load("en_core_web_sm")
  doc_terms = nlp_en(text)
  terms = [ent.text for ent in doc_terms.ents]

  nlp = English()
  retokenizer = MatchRetokenizeComponent(nlp, terms)
  nlp.add_pipe(retokenizer, name='merge_phrases', last=True)

  doc = nlp(text)
  return [tok for tok in doc]
```

# PyTorch 모델 구현 - Dataset

*Sequence to Sequence Learning with Neural Networks*

## PyTorch에 들어가기 위한 데이터셋 생성

Field(=파이프라인)를 통해
각 언어별 토큰화 &
앞, 뒤에 <sos>, <eos> 삽입

```
SRC = Field(tokenize = tokenize_kor, #
            init_token = '<sos>',
            eos_token = '<eos>',
            lower = True)

TRG = Field(tokenize = tokenize_en,
            init_token = '<sos>',
            eos_token = '<eos>',
            lower = True)
```

TabularDataset(=PyTorch에서 사용하는 형태의 데이터셋)으로 형변환

```
train_data, valid_data, test_data = TabularDataset.splits(path='.',
        train='/content/drive/My Drive/Seq to Seq/train_df.tsv',
        validation='/content/drive/My Drive/Seq to Seq/val_df.tsv',
        test='/content/drive/My Drive/Seq to Seq/test_df.tsv',
        format='tsv',
        fields=[('src', SRC), ('trg', TRG)], skip_header=True)
```

# PyTorch 모델 구현 - Dataset

*Sequence to Sequence Learning with Neural Networks*

## PyTorch에 들어가기 위한 데이터셋 기본 설정

```
1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') #GPU 쓰라고 하는 명령
```

```
1 BATCH_SIZE = 128
```

```
1 train_iterator  = BucketIterator(#비슷한 길이의 문장끼리 하나의 배치로 묶어주라는 함수
2     train_data,
3     batch_size = BATCH_SIZE,
4     device = device)
5
```

# PyTorch 모델 구현 - Encoder

*Sequence to Sequence Learning with Neural Networks*

## Encoder 클래스 생성

단어 임베딩

드랍아웃 적용

RNN(LSTM)을 적용

```python
class Encoder(nn.Module):#임베
    def __init__(self, input_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()

        self.hid_dim = hid_dim
        self.n_layers = n_layers

        self.embedding = nn.Embedding(input_dim, emb_dim)

        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout = dropout)

        self.dropout = nn.Dropout(dropout)

    def forward(self, src):

        #src = [src len, batch size]

        embedded = self.dropout(self.embedding(src))

        #embedded = [src len, batch size, emb dim]

        outputs, (hidden, cell) = self.rnn(embedded)#히든, 셀을 받아줌

        #outputs = [src len, batch size, hid dim * n directions]
        #hidden = [n layers * n directions, batch size, hid dim]
        #cell = [n layers * n directions, batch size, hid dim]

        #outputs are always from the top hidden layer

        return hidden, cell#히든셀이 출력됨
```

# PyTorch 모델 구현 - Decoder

*Sequence to Sequence Learning with Neural Networks*

## Decoder 클래스 생성

Encoder에서 출력된
배치사이즈, 히든, 셀을 입력

RNN(LSTM)을 적용해
디코딩

```python
class Decoder(nn.Module):
    def __init__(self, output_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()

        self.output_dim = output_dim
        self.hid_dim = hid_dim
        self.n_layers = n_layers

        self.embedding = nn.Embedding(output_dim, emb_dim)

        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout = dropout)

        self.fc_out = nn.Linear(hid_dim, output_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, cell):

        input = input.unsqueeze(0)#0번째 자리에 새로운 차원 삽입

        #input = [1, batch size]

        embedded = self.dropout(self.embedding(input))

        #embedded = [1, batch size, emb dim]

        output, (hidden, cell) = self.rnn(embedded, (hidden, cell))

        prediction = self.fc_out(output.squeeze(0))#예측한 문장 출력


        return prediction, hidden, cell
```

# PyTorch 모델 구현 - Seq2Seq Class

*Sequence to Sequence Learning with Neural Networks*

## Seq2seq 클래스 생성

인코더, 디코더의
차원과 레이어를 통일

Teacher_forcing 활용

```python
class Seq2Seq(nn.Module):#인코더와 디코더, gpu활용
    def __init__(self, encoder, decoder, device):
        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.device = device

        assert encoder.hid_dim == decoder.hid_dim, ₩
            "Hidden dimensions of encoder and decoder must be equal!"
        assert encoder.n_layers == decoder.n_layers, ₩
            "Encoder and decoder must have equal number of layers!"
    def forward(self, src, trg, teacher_forcing_ratio = 0.5):

        batch_size = trg.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.output_dim

        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.device)

        hidden, cell = self.encoder(src)
        input = trg[0,:]

        for t in range(1, trg_len):
            output, hidden, cell = self.decoder(input, hidden, cell)
            outputs[t] = output

            teacher_force = random.random() < teacher_forcing_ratio

            top1 = output.argmax(1) #가장 높은 확률 예측단어(문장) 뽑아내기
            input = trg[t] if teacher_force else top1
        return outputs
```

# PyTorch 모델 구현 - Seq2Seq Modeling

*Sequence to Sequence Learning with Neural Networks*

## Seq2seq 모델 생성(2)

### 하이퍼 파라미터 튜닝

```python
INPUT_DIM = len(SRC.vocab)
OUTPUT_DIM = len(TRG.vocab)
ENC_EMB_DIM = 256
DEC_EMB_DIM = 256
HID_DIM = 512
N_LAYERS = 2
ENC_DROPOUT = 0.5
DEC_DROPOUT = 0.5

enc = Encoder(INPUT_DIM, ENC_EMB_DIM, HID_DIM, N_LAYERS, ENC_DROPOUT)
dec = Decoder(OUTPUT_DIM, DEC_EMB_DIM, HID_DIM, N_LAYERS, DEC_DROPOUT)

model = Seq2Seq(enc, dec, device).to(device)
```

```python
def init_weights(m):
    for name, param in m.named_parameters():
        nn.init.uniform_(param.data, -0.08, 0.08)
        #논문처럼 파라미터를 -0.08과 0.08사이로 맞춰주기

model.apply(init_weights)
```

```python
optimizer = optim.Adam(model.parameters())#아담으로 하지말고 SGD로 해보자
```

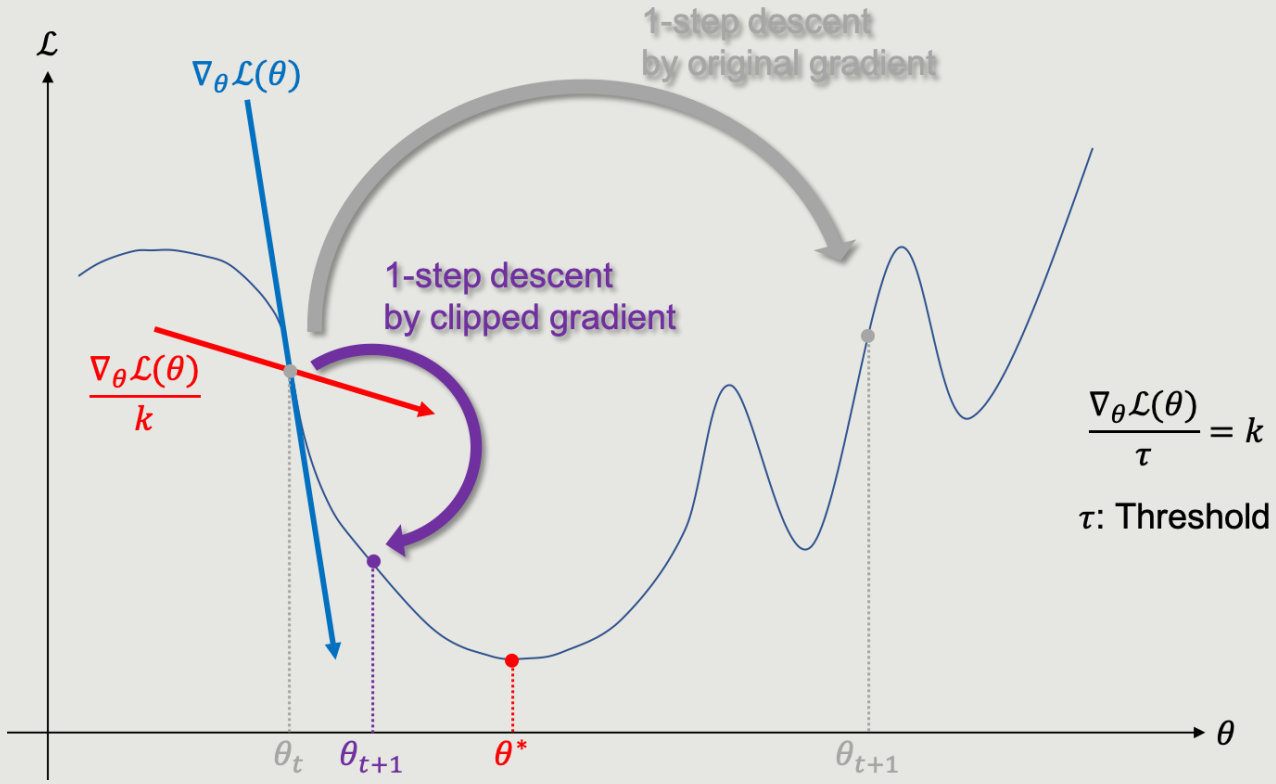
```python
TRG_PAD_IDX = TRG.vocab.stoi[TRG.pad_token]#타겟(영어) 패딩해주고

criterion = nn.CrossEntropyLoss(ignore_index = TRG_PAD_IDX)
#기준으로 삼아서 트레인 채점할 때 사용
```

# PyTorch 모델 구현 - Seq2Seq Modeling

*Sequence to Sequence Learning with Neural Networks*

## Seq2seq train 모델 생성

### Teacher_forcing을 활용,
### Gradient clipping 사용




```python
def train(model, iterator, optimizer, criterion, clip):# 테스트 후 loss를 리턴

    model.train()

    epoch_loss = 0

    for i, batch in enumerate(iterator):

        src = batch.src
        trg = batch.trg

        optimizer.zero_grad()#gradient 초기화

        output = model(src, trg)

        #trg = [trg len, batch size]
        #output = [trg len, batch size, output dim]

        output_dim = output.shape[-1]

        output = output[1:].view(-1, output_dim)
        trg = trg[1:].view(-1)

        #trg = [(trg len - 1) * batch size]
        #output = [(trg len - 1) * batch size, output dim]

        loss = criterion(output, trg)#타겟과아웃풋 비교후 로스로 입력

        loss.backward()#역전파 계산

        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        #gradient clipping->norm을 수정해가면서 발산 막아주기->논문방
        #아래 링크 사진확인
        #https://kh-kim.gitbook.io/natural-language-processing-with-pytorch/00-cover-6/

        optimizer.step()

        epoch_loss += loss.item()

    return epoch_loss / len(iterator)
```

# PyTorch 모델 구현 - Seq2Seq Model

*Sequence to Sequence Learning with Neural Networks*

## Seq2seq 평가 모델 생성

Teacher_forcing 미활용

```python
def evaluate(model, iterator, criterion):#평가 함수, 여기선 teacher forcing안쓰고

    model.eval()

    epoch_loss = 0

    with torch.no_grad():

        for i, batch in enumerate(iterator):

            src = batch.src
            trg = batch.trg

            output = model(src, trg, 0) #turn off teacher forcing

            #trg = [trg len, batch size]
            #output = [trg len, batch size, output dim]

            output_dim = output.shape[-1]

            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)

            #trg = [(trg len - 1) * batch size]
            #output = [(trg len - 1) * batch size, output dim]

            loss = criterion(output, trg)

            epoch_loss += loss.item()

    return epoch_loss / len(iterator)
```

# PyTorch 모델 구현 - Seq2Seq train

*Sequence to Sequence Learning with Neural Networks*

## Seq2seq 모델 훈련

```
N_EPOCHS = 20
CLIP = 5#논문처럼 5로 맞춰줌. SGD가 적용되지 않아 큰 영향을 주지는 못할 듯.

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss = train(model, train_iterator, optimizer, criterion, CLIP)
    valid_loss = evaluate(model, valid_iterator, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut1-model.pt')

    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train PPL: {math.exp(train_loss):7.3f}')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. PPL: {math.exp(valid_loss):7.3f}')
```

# PyTorch 모델 구현 - Seq2Seq test

*Sequence to Sequence Learning with Neural Networks*

## Seq2seq test 모델 실행

```
1 model.load_state_dict(torch.load('tut1-model.pt'))
2
3 test_loss = evaluate(model, test_iterator, criterion)
4
5 print(f'| Test Loss: {test_loss:.3f} | Test PPL: {math.exp(test_loss):7.3f} |')
```

PPL: 간접광고X, Perplexity. 주사위로 생각하면 한번 던질 때 나올 경우의 수로 생각할 수 있다.

▶ 얼마나 많은 단어의 개수를 가지고 정답일지 고민하는지의 개수.

# PyTorch 모델 구현 - Seq2Seq train

*Sequence to Sequence Learning with Neural Networks*

## Seq2seq 결과

### 독일어-영어(예제 데이터)

```
Epoch: 15 | Time: 0m 37s
        Train Loss: 2.755 | Train PPL:  15.727
         Val. Loss: 3.746 |  Val. PPL:  42.342
```

```
| Test Loss: 3.705 | Test PPL:  40.644 |
```

### 한국어-영어(역방향)

```
Epoch: 15 | Time: 0m 50s
        Train Loss: 2.575 | Train PPL:  13.135
         Val. Loss: 4.623 |  Val. PPL: 101.749
```

```
| Test Loss: 4.564 | Test PPL:  95.946 |
```

### 한국어-영어(정방향)

```
Epoch: 15 | Time: 1m 36s
        Train Loss: 2.919 | Train PPL:  18.517
         Val. Loss: 4.548 |  Val. PPL:  94.488
```

```
| Test Loss: 4.558 | Test PPL:  95.394 |
```

# PyTorch 모델 구현 - 결론

*Sequence to Sequence Learning with Neural Networks*

## Generating sequences with recurrent neural networks(A. Graves, 2013)

## WMT’ 14 English to French MT task

# PyTorch 모델 구현 - 결론

*Sequence to Sequence Learning with Neural Networks*

## 논문에서 구현하고자 하는 기능을 이미 함수로 설정

▶따로 For문, If등을 써서 설정할 필요 X
▶사용법에 익숙해지면 더욱 다양하고 섬세한 Hyper-parameter 튜닝가능

## Model.predict 방법을 for문을 이용한 함수를 구현해야 했기에 번역결과를 보지 못한점이 아쉬움.

# Keras 모델 구현 - Input 역순 배열

*Sequence to Sequence Learning with Neural Networks*

```
kor_eng['ko']=kor_eng['ko'].apply(lambda x: '.'+(' '.join(x.replace('.','').split(' ')[::-1])))
```

```
kor_eng['ko']
```

```
0          .가요 다락방으로 만나러 배트를 저녁 매일 나는
1                  .가요 안 이해가 이문장이 선생님
2            .가요 빠르게 너무 시간이 시작하면 컴퓨터를
3               .가요 돌아 한국으로 자정에 오늘 나는
4                  .가요 화장실에 일어나자마자 나는
                        ...
74995        .힘들어 통학하기 멀어서 학교가 고민은 나의
74996           .힘들어 충분히 내고양이때문에 지금 난
74997          .힘들어? 많이 것이 어려운 대화가 나와
74998          .힘들어? 그렇게 연락하는게 한번 하루에
74999      .힘들죠 많이 즐기기엔 스포츠를 아이들이 어린
Name: ko, Length: 75000, dtype: object
```

```
kor_eng_corp=(kor_eng['ko']+'₩t'+kor_eng['en']).tolist()
```

```
ko, en = create_dataset(kor_eng_corp, None)
print(ko[-1])
print(en[-1])
```

```
<start> . 힘들죠 많이 즐기기엔 스포츠를 아이들이 어린 <end>
<start> it is difficult for young children to enjoy sports . <end>
```

## Input Data 역순 배열

- [::-1]을 통해 split된 단어들을 역순으로 배열

**. 가 역순 배열의 첫 단어와 두 번째 단어의 사이로 위치하는 문제 발생**

**- replace로 .을 제거 한 후 다시 추가**

significantly outperformed shallow LSTMs, so we chose an LSTM with four layers. Third, we found it extremely valuable to reverse the order of the words of the input sentence. So for example, instead of mapping the sentence $a, b, c$ to the sentence $\alpha, \beta, \gamma$, the LSTM is asked to map $c, b, a$ to $\alpha, \beta, \gamma$, where $\alpha, \beta, \gamma$ is the translation of $a, b, c$. This way, $a$ is in close proximity to $\alpha$, $b$ is fairly close to

▲ *입력 문장의 단어 배열을 뒤집는 것이 효과적*

# Keras 모델 구현 - 토크나이징 & Train_test split

*Sequence to Sequence Learning with Neural Networks*

## 토크나이징

- Keras 의 Tokenizer를 사용해 Input Language(한글)와 Target Language(영어)를 토크나이징

## Train_test_split

- Scikit Learn의 train_test_split을 사용해 75000개의 데이터를 80%의 Train_set과 20%의 Test_set으로 분리

```
[ ]  # 토크나이저
     def tokenize(lang):
       lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
           filters='')
       lang_tokenizer.fit_on_texts(lang)
       tensor = lang_tokenizer.texts_to_sequences(lang)
       tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                              padding='post')
       return tensor, lang_tokenizer
```

```
[ ]   def load_dataset(path, num_examples=None):
       # creating cleaned input, output pairs
       inp_lang, targ_lang = create_dataset(path, num_examples)
       input_tensor, inp_lang_tokenizer = tokenize(inp_lang)
       target_tensor, targ_lang_tokenizer = tokenize(targ_lang)
       return input_tensor, target_tensor, inp_lang_tokenizer, targ_lang_tokenizer
```

```
[ ]  # Try experimenting with the size of that dataset
     num_examples = 63000
     input_tensor, target_tensor, inp_lang, targ_lang = load_dataset(kor_eng_corp, num_examples)

     # Calculate max_length of the target tensors
     max_length_targ, max_length_inp = target_tensor.shape[1], input_tensor.shape[1]
```

```
[ ]   # Creating training and validation sets using an 80-20 split
     input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val = train_test_split(input_tensor, target_tensor, test_size=0.20633334)

     # Show length
     print(len(input_tensor_train), len(target_tensor_train), len(input_tensor_val), len(target_tensor_val))
```

```
50000 50000 13000 13000
```

```
print ("Input Language; index to word mapping")
convert(inp_lang, input_tensor_train[0])
print ()
print ("Target Language; index to word mapping")
convert(targ_lang, target_tensor_train[0])
```

```
Input Language; index to word mapping
1 ----> <start>
2 ----> .
7 ----> 있어
6 ----> 수
2675 ----> 읽을
9893 ----> 책도
347 ----> 가면
35363 ----> 정보실에
35364 ----> 문헌
14937 ----> 공부하다가
3 ----> <end>

Target Language; index to word mapping
1 ----> <start>
187 ----> while
4 ----> i
190 ----> study
10 ----> ,
4 ----> i
90 ----> could
50 ----> go
15 ----> and
375 ----> read
638 ----> books
3 ----> .
2 ----> <end>
```

# Keras 모델 구현 - Encoder

*Sequence to Sequence Learning with Neural Networks*

```
class Encoder(tf.keras.Model):
 def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
   super(Encoder, self).__init__()
   self.batch_sz = batch_sz
   self.enc_units = enc_units
   self.initializer = tf.keras.initializers.RandomUniform(minval=-0.08, maxval=0.08, seed=None)
   self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
   self.
                                  recurrent_initializer=self.initializer))

 def call(self, x, hidden):
   x = self.embedding(x)
   output, state_fh, state_fc, state_bh, state_bc = self.lstm(x, initial_state = hidden)
   output, state_fh, state_fc, state_bh, state_bc = self.lstm(x, initial_state=[state_fh, state_fc, state_bh, state_bc])
   output, state_fh, state_fc, state_bh, state_bc = self.lstm(x, initial_state=[state_fh, state_fc, state_bh, state_bc])
   # output, state_fh, state_fc, state_bh, state_bc = self.lstm(x, initial_state=[state_fh, state_fc, state_bh, state_bc])
   # state_h = Concatenate()([state_fh, state_bh])
   # state_c = Concatenate()([state_fc, state_bc])
   return output, state_fh, state_fc, state_bh, state_bc

 def initialize_hidden_state(self):
   return [tf.zeros((self.batch_sz, self.enc_units)) for i in range(4)]
```

- We initialized all of the LSTM's parameters with the uniform distribution between -0.08 and 0.08

```
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)
```

```
# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden_fh, sample_hidden_fc, sample_hidden_bh, sample_hidden_bc = encoder(example_input_batch, sample_hidden)
print ('Encoder output shape: (batch size, sequence length, units) {}'.format(sample_output.shape))
print ('Encoder Hidden state FH shape: (batch size, units) {}'.format(sample_hidden_fh.shape))
print ('Encoder Hidden state FC shape: (batch size, units) {}'.format(sample_hidden_fc.shape))
print ('Encoder Hidden state BH shape: (batch size, units) {}'.format(sample_hidden_bh.shape))
print ('Encoder Hidden state BC shape: (batch size, units) {}'.format(sample_hidden_bc.shape))
```

```
Encoder output shape: (batch size, sequence length, units) (64, 21, 2048)
Encoder Hidden state FH shape: (batch size, units) (64, 1024)
Encoder Hidden state FC shape: (batch size, units) (64, 1024)
Encoder Hidden state BH shape: (batch size, units) (64, 1024)
Encoder Hidden state BC shape: (batch size, units) (64, 1024)
```

## LSTM layer

- __Init__에서 Lstm layer를 쌓는 것이 아니라 call에서 layer를 쌓는다.
- def(call)에서 첫번째 layer의 output, 정방향 은닉층과 셀, 역방향 은닉층과 셀을 그 다음 layer에 전달
- *- state_fh, state_fc 는 정방향 은닉층과 그 때의 셀,*
- *- state_bh, state_bc 는 역방향 은닉층과 그 때의 셀*

## Bidirection RNN

*"나는 __를 뒤집어 쓰고 펑펑 울었다."*
에서 '나는' 뒤에 나올 수 있는 단어는 많지만
'뒤집어 쓰고 펑펑 울었다' 앞에 쓰일 수 있는 단어는
한정적

# Keras 모델 구현 - 모델 세부 설정

*Sequence to Sequence Learning with Neural Networks*

**배치 사이즈 = 64**
**임베딩 차원 = 1000**
**Vocab_inp_size = 85239**
**Vocab_tar_size = 19231**

```
BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 64
steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
embedding_dim = 1000
units = 1024
vocab_inp_size = len(inp_lang.word_index)+1
vocab_tar_size = len(targ_lang.word_index)+1

dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train, target_tensor_train)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
```

## BUFFER_SIZE & BATCH_SIZE

- BUFFER_SIZE는 SHUFFLE을 하기 위해 필요한 공간의 크기
- BUFFER_SIZE는 데이터 SHUFFLE 시 설정된 BUFFER_SIZE만큼의 데이터를 SHUFFLE하게 되고, BATCH_SIZE만큼의 데이터를 한번에 학습한다.

# Keras 모델 구현 - Bahdanau Attention

*Sequence to Sequence Learning with Neural Networks*

```
class BahdanauAttention(tf.keras.layers.Layer):
 def __init__(self, units):
   super(BahdanauAttention, self).__init__()
   self.W1 = tf.keras.layers.Dense(units)
   self.W2 = tf.keras.layers.Dense(units)
   self.V = tf.keras.layers.Dense(1)

 def call(self, query1, query2, values):
   # query hidden state shape == (batch_size, hidden size)
   # query_with_time_axis shape == (batch_size, 1, hidden size)
   # values shape == (batch_size, max_len, hidden size)
   # we are doing this to broadcast addition along the time axis to calculate the score
   query=Concatenate()([query1, query2])
   query_with_time_axis = tf.expand_dims(query, 1)

   # score shape == (batch_size, max_length, 1)
   # we get 1 at the last axis because we are applying score to self.V
   # the shape of the tensor before applying self.V is (batch_size, max_length, units)
   score = self.V(tf.nn.tanh(
       self.W1(query_with_time_axis) + self.W2(values)))

   # attention_weights shape == (batch_size, max_length, 1)
   attention_weights = tf.nn.softmax(score, axis=1)

   # context_vector shape after sum == (batch_size, hidden_size)
   context_vector = attention_weights * values
   context_vector = tf.reduce_sum(context_vector, axis=1)

   return context_vector, attention_weights
```

```
attention_layer = BahdanauAttention(10)
attention_result, attention_weights = attention_layer(sample_hidden_fh, sample_hidden_bh, sample_output)

print("Attention result shape: (batch size, units) {}".format(attention_result.shape))
print("Attention weights shape: (batch_size, sequence_length, 1) {}".format(attention_weights.shape))
```

## 기존 Encoder_Decoder의 단점

- 소스 문장을 고정된 길이의 벡터로 인코딩
- 문장이 길어질수록 더 많은 길이를 고정된 길이로 압축하는 과정에서 정보의 손실이 발생

## Bahdanau Attention

- 벡터들의 Sequence로 인코딩
  ▶ 고정된 길이의 문장 임베딩이 필요 하지 않음

- **Attention 이전**
  ▶ 디코더가 전 단계 출력의 단어와, 전 단계의 Hidden_state 만을 활용하여 계산

- **Attention 활용**
  ▶ 전 단계 출력의 단어와 전 단계의 Hidden_state + Context vector를 함께 활용하여 계산

# Keras 모델 구현 - Decoder

*Sequence to Sequence Learning with Neural Networks*

```python
class Decoder(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
    super(Decoder, self).__init__()
    self.batch_sz = batch_sz
    self.dec_units = dec_units
    self.initializer=tf.keras.initializers.RandomUniform(minval=-0.08, maxval=0.08, seed=None)
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.lstm = tf.keras.layers.Bidirectional(LSTM(self.dec_units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer=self.initializer))
    self.fc = tf.keras.layers.Dense(vocab_size)

    # used for attention
    self.attention = BahdanauAttention(self.dec_units)

  def call(self, x, hidden_fh, hidden_fc, hidden_bh, hidden_bc, enc_output):
    # enc_output shape == (batch_size, max_length, hidden_size)
    context_vector, attention_weights = self.attention(hidden_fh, hidden_bh, enc_output)

    # x shape after passing through embedding == (batch_size, 1, embedding_dim)
    x = self.embedding(x)

    # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state_fh, state_fc, state_bh, state_bc = self.lstm(x, initial_state=[hidden_fh, hidden_fc, hidden_bh, hidden_bc])
    output, state_fh, state_fc, state_bh, state_bc = self.lstm(x, initial_state=[state_fh, state_fc, state_bh, state_bc])
    output, state_fh, state_fc, state_bh, state_bc = self.lstm(x, initial_state=[state_fh, state_fc, state_bh, state_bc])
    # output, state_fh, state_fc, state_bh, state_bc = self.lstm(x, initial_state=[state_fh, state_fc, state_bh, state_bc])
    # state_h = Concatenate()([state_fh, state_bh])
    # state_c = Concatenate()([state_fc, state_bc])

    # output shape == (batch_size * 1, hidden_size)
    output = tf.reshape(output, (-1, output.shape[2]))

    # output shape == (batch_size, vocab)
    x = self.fc(output)

    return x, state_fh, state_fc, state_bh, state_bc, attention_weights
```

## 디코더

인코더의 은닉상태와
*#정방향 은닉층과 셀의 값,*
*#역방향 은닉층과 셀의 값,*

Attention을 통해 얻은
context vector를

함께 디코딩에 사용

# Keras 모델 구현 - Optimizer

*Sequence to Sequence Learning with Neural Networks*

```
optimizer = tf.keras.optimizers.SGD(lr=0.7)
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_mean(loss_)
```

## Optimizer = SGD

- 논문에서 사용한 SGD Optimizer를 사용
- Learning rate 또한 논문과 마찬가지로 0.7을 사용

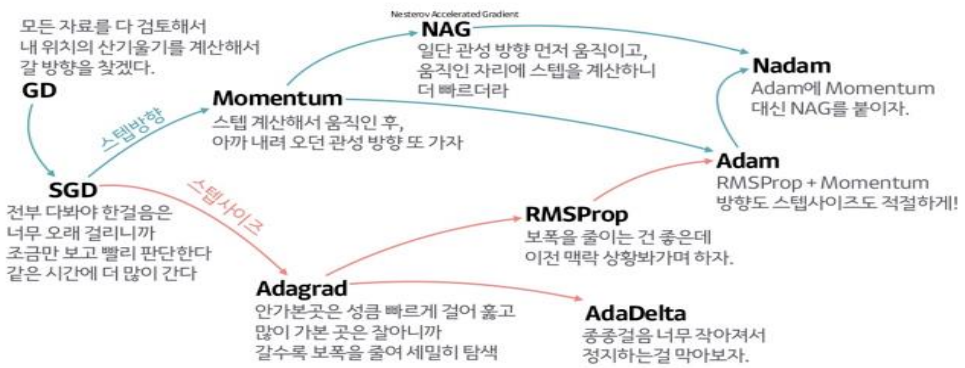
## SGD vs ADAM

### SGD

Mini batch 사이즈만큼 돌려서 최적의 값을 찾는 과정, 빠르지만 방향설정이 뒤죽박죽이며, 적절한 learning rate를 조절이 어려움

### ADAM

SGD의 단점인 learning rate와 방향 모두를 고려한 Optimizer



# Keras 모델 구현 - Seq2Seq train

*Sequence to Sequence Learning with Neural Networks*

```
@tf.function
def train_step(inp, targ, enc_hidden):
  loss = 0

  with tf.GradientTape() as tape:
    enc_output, enc_hidden_fh, enc_hidden_fc, enc_hidden_bh, enc_hidden_bc = encoder(inp, enc_hidden)

    dec_hidden_fh = enc_hidden_fh
    dec_hidden_fc = enc_hidden_fc
    dec_hidden_bh = enc_hidden_bh
    dec_hidden_bc = enc_hidden_bc

    dec_input = tf.expand_dims([targ_lang.word_index['<start>']] * BATCH_SIZE, 1)

    # Teacher forcing - feeding the target as the next input
    for t in range(1, targ.shape[1]):
      # passing enc_output to the decoder
      predictions, dec_hidden_fh,dec_hidden_fc, hidden_bh, hidden_bc, _ = decoder(dec_input, dec_hidden_fh,dec_hidden_fc, dec_hidden_bh, dec_hidden_bc, enc_output)

      loss += loss_function(targ[:, t], predictions)

      # using teacher forcing
      dec_input = tf.expand_dims(targ[:, t], 1)

  batch_loss = (loss / int(targ.shape[1]))

  variables = encoder.trainable_variables + decoder.trainable_variables

  gradients = tape.gradient(loss, variables)
  # print(gradients)
  # lower_g, higher_g
  # g=gradients/128
  # s=numpy.linalg.norm(g, ord=2)
  # if s > 5:
  #   gradients= (5+g)/s

  optimizer.apply_gradients(zip(gradients, variables))

  return batch_loss
```

## Tf.GradientTape

- 컨텍스트 안에서 실행된 모든 연산을 tape에 기록
- 이후 후진 방식 자동 미분을 사용해 tape에 기록된 연산의 Gradient를 계산

## Teacher Forcing

- 이전 시점의 디코더 셀의 출력이 아닌
  이전 시점의 실제 값을 현재 시점의 입력으로 받는다

이전 시점의 디코더 셀의 예측이 틀린 상태에서
현재 시점의 디코더 셀의 입력으로 사용하면
현재 시점의 디코더 셀의 예측도 잘못될 가능성이 발생

# Keras 모델 구현 - Epochs

*Sequence to Sequence Learning with Neural Networks*

```
EPOCHS = 7
learning_count = 0

for epoch in range(EPOCHS):
  start = time.time()
  learning_count += 1
  enc_hidden = encoder.initialize_hidden_state()
  total_loss = 0
  if learning_count <= 5:
    for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):
      batch_loss = train_step(inp, targ, enc_hidden)
      total_loss += batch_loss
      if batch % 100 == 0:
        print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
                                                     batch,
                                                     batch_loss.numpy()))
    # saving (checkpoint) the model every 2 epochs
    # if (epoch + 1) % 2 == 0:
    #   checkpoint.save(file_prefix = checkpoint_prefix)
    print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                                        total_loss / steps_per_epoch))
    print('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
  else:
    for_lr = learning_count - 5
    optimizer = tf.keras.optimizers.SGD(learning_rate=0.7*np.power(0.5, for_lr))
    for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):
      batch_loss = train_step(inp, targ, enc_hidden)
      total_loss += batch_loss
      if batch % 100 == 0:
        print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
                                                     batch,
                                                     batch_loss.numpy()))
    # # saving (checkpoint) the model every 2 epochs
    # if (epoch + 1) % 2 == 0:
    #   checkpoint.save(file_prefix = checkpoint_prefix)
    print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                                        total_loss / steps_per_epoch))
    print('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
```

- We used stochastic gradient descent without momentum, with a fixed learning rate of 0.7. After 5 epochs, we begun halving the learning rate every half epoch. We trained our models for a total of 7.5 epochs.

## Learning rate 조절 (#시도)

- learning_count를 초기 0으로 설정

- epoch 마다 learning_count에 1을 더한 뒤 Learning_count가 5를 넘어서면 learning_rate를 0.7에서 반으로 줄도록 설정하려함

- 하지만 알 수 없는 이유로 실패

# Keras 모델 구현 - Evaluate

*Sequence to Sequence Learning with Neural Networks*

```
def evaluate(sentence):
  attention_plot = np.zeros((max_length_targ, max_length_inp))

  sentence = preprocess_sentence(sentence)

  inputs = [inp_lang.word_index[i] for i in sentence.split(' ')]
  inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
                                                         maxlen=max_length_inp,
                                                         padding='post')
  inputs = tf.convert_to_tensor(inputs)

  result = ''

  hidden = [tf.zeros((1, units)) for i in range(4)]
  enc_output, enc_hidden_fh, enc_hidden_fc, enc_hidden_bh, enc_hidden_bc = encoder(inputs, hidden)

  dec_hidden_fh = enc_hidden_fh
  dec_hidden_fc = enc_hidden_fc
  dec_hidden_bh = enc_hidden_bh
  dec_hidden_bc = enc_hidden_bc

  dec_input = tf.expand_dims([targ_lang.word_index['<start>']], 0)

  for t in range(max_length_targ):
    predictions, dec_hidden_fh, dec_hidden_fc, dec_hidden_bh, dec_hidden_bc, attention_weights = decoder(dec_input,
                                                                                                         dec_hidden_fh,
                                                                                                         dec_hidden_fc,
                                                                                                         dec_hidden_bh,
                                                                                                         dec_hidden_bc,
                                                                                                         enc_output)

    # storing the attention weights to plot later on
    attention_weights = tf.reshape(attention_weights, (-1, ))
    attention_plot[t] = attention_weights.numpy()

    predicted_id = tf.argmax(predictions[0]).numpy()

    result += targ_lang.index_word[predicted_id] + ' '

    if targ_lang.index_word[predicted_id] == '<end>':
      return result, sentence, attention_plot

    # the predicted ID is fed back into the model
    dec_input = tf.expand_dims([predicted_id], 0)

  return result, sentence, attention_plot
```

## 평가

1. 새로운 데이터를 인코더의 Input의 형태로 변환
- 단어의 정수 인덱스 배열
- 지정한 길이만큼의 Padding 삽입

2. 해당 Input을 Encoder에 입력 이후 출력된 Hidden_state(은닉층, 셀)을 Decoder의 Input 형태로 변환

3. Decoder에서 예측값과 Hidden_state, Context Vector를 계산하여 다음 Decoder에 입력값으로 전달

4. 해당 출력 예측값이 <end>일 경우 연산을 중단

# Keras 모델 구현 - BLEU Score

*Sequence to Sequence Learning with Neural Networks*

## BLEU SCORE 계산

### TEST 문장에 단어가 없는 경우

### ▶TRY, EXCEPT로 해결

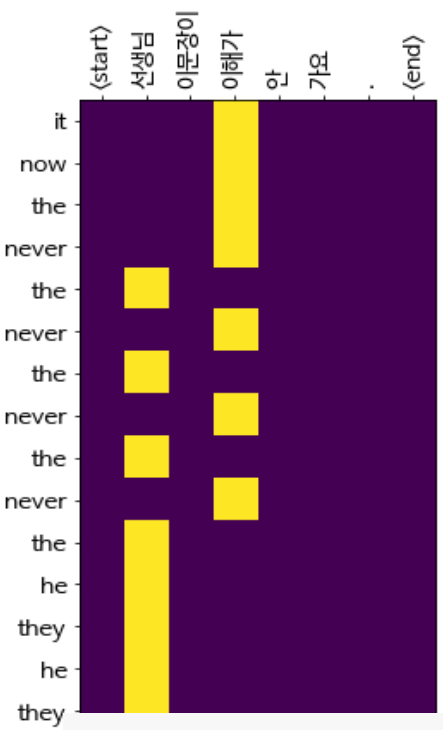
```
import nltk
import nltk.translate.bleu_score as bleu
from nltk.translate.bleu_score import SmoothingFunction

def bleu_avg(doc, target_doc):
  bleu_score=[]
  smoothie = SmoothingFunction().method4
  for input_sentence, target_sentence in tqdm(zip(doc,target_doc)):
    try:
      input_sentence=re.sub('<[^>]*>','',input_sentence)
      target_sentence=re.sub('<[^>]*>','',target_sentence)
      result, sentence, attention_plot = evaluate(input_sentence)
      bleu_score.append(nltk.translate.bleu_score.sentence_bleu(target_sentence.split(' '),
                                                                result.split(' '),smoothing_function=smoothie))
    except:
      pass
  return np.mean(bleu_score)
```

# Keras 모델 구현 - BIDIRECTIONAL LSTM Model (1 layer)

***Sequence to Sequence Learning with Neural Networks***

```
Input: <start> 나는 매일 저녁 배트를 만나러 다락방으로 가요 . <end>
Predicted translation: it even the the the the the the the he he he . . . ! ! ! ! !
```


```
Input: <start> 선생님 이문장이 이해가 안 가요 . <end>
Predicted translation: it now the never the never the never the never the he they he they . . . . ! !
```




```
# right order sentece, 1 layers Bidirectional LSTM with 1000 embedding_dims
bleu_avg(ko[63001:],en[63001:])
```

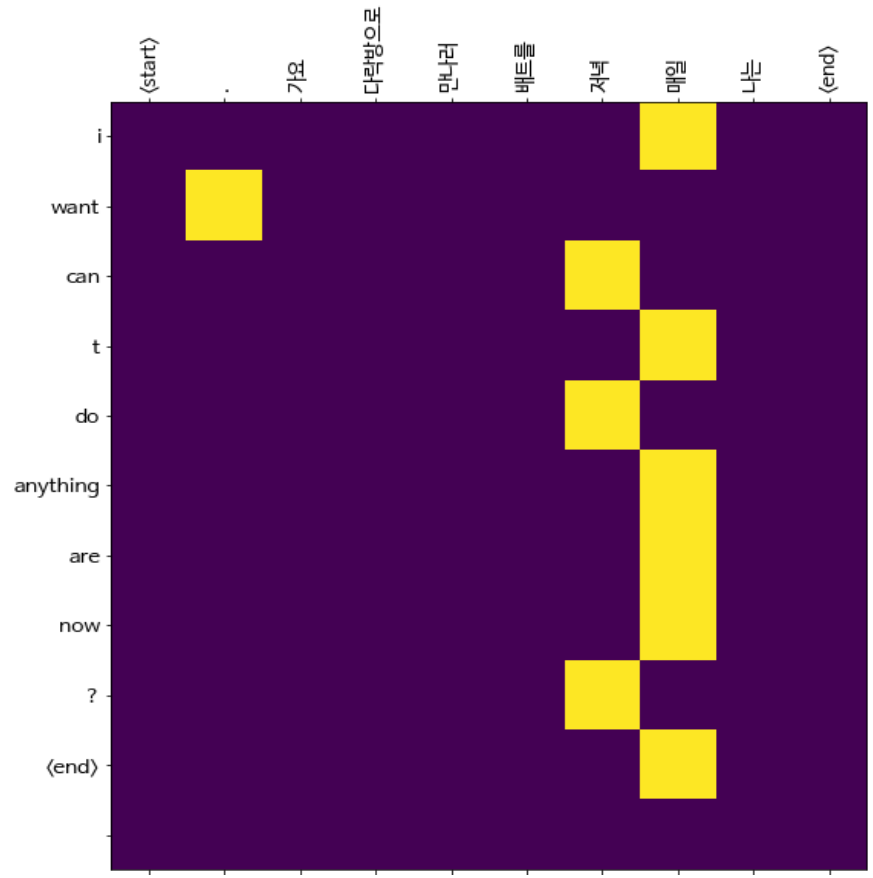
```
11999/? [12:03<00:00, 16.59it/s]
```

```
0.17252454503274248
```

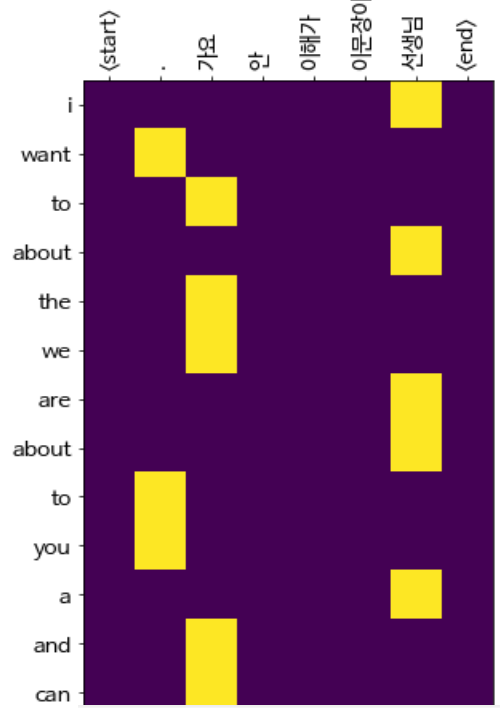
# Keras 모델 구현 - BIDIRECTIONAL LSTM reverse(1 layer)

***Sequence to Sequence Learning with Neural Networks***

```
Input: <start> . 가요 다락방으로 만나러 배트를 저녁 매일 나는 <end>
Predicted translation: i want can t do anything are now ? <end>
```




```
Input: <start> . 가요 안 이해가 이문장이 선생님 <end>
Predicted translation: i want to about the we are about to you a and can i ? <end>
```



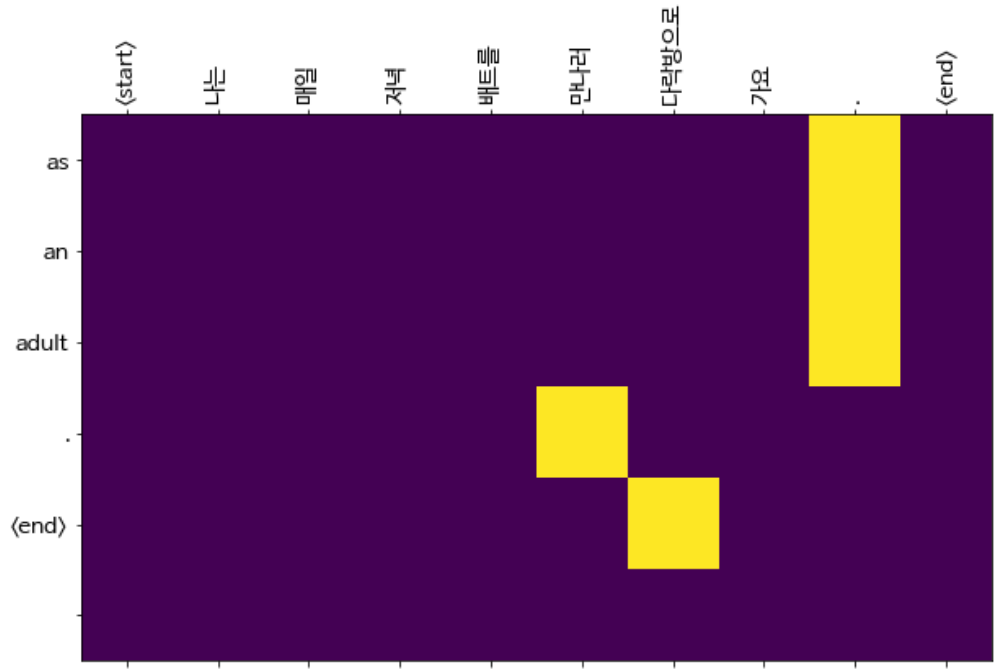

```
# reverse order sentece, 1 layers Bidirectional LSTM with 1000 embedding_dims
bleu_avg(ko[63001:],en[63001:])
```

```
11999/? [08:20<00:00, 23.95it/s]

0.19913371436160368
```

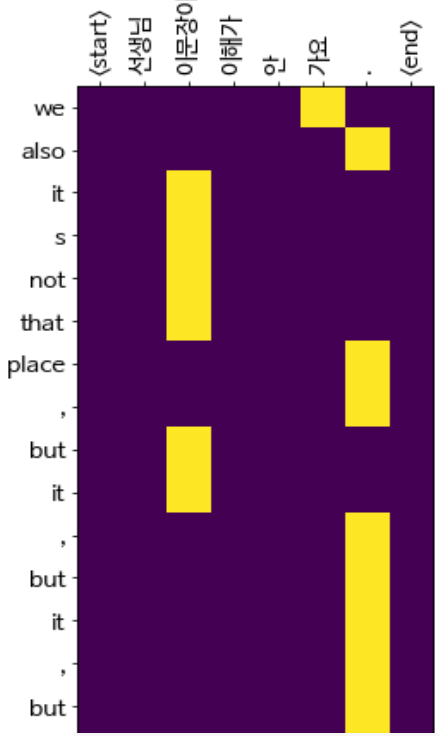
# Keras 모델 구현 - BIDIRECTIONAL LSTM Model (2 layers)

***Sequence to Sequence Learning with Neural Networks***

```
Input: <start> 나는 매일 저녁 배트를 만나러 다락방으로 가요 . <end>
Predicted translation: as an adult . <end>
```




```
Input: <start> 선생님 이문장이 이해가 안 가요 . <end>
Predicted translation: we also it s not that place , but it , but it , but it , but
```



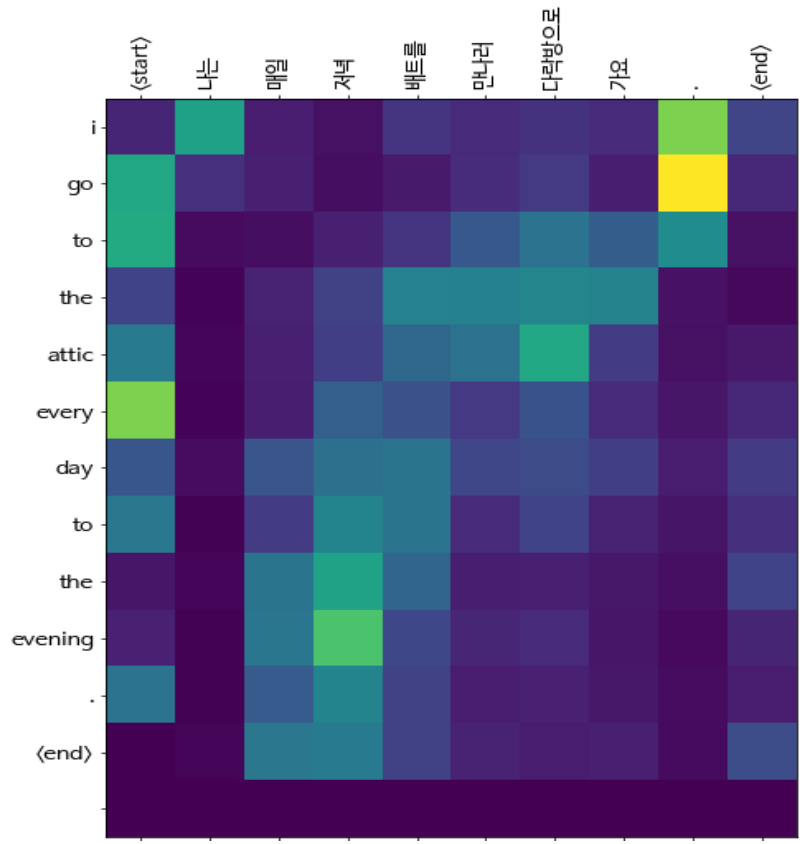

```
[37] # right order sentece, 2 layers Bidirectional LSTM with 1000 embedding_dims
     bleu_avg(ko[63001:],en[63001:])
```

```
11999/? [04:37<00:00, 43.17it/s]

0.22938764631821015
```

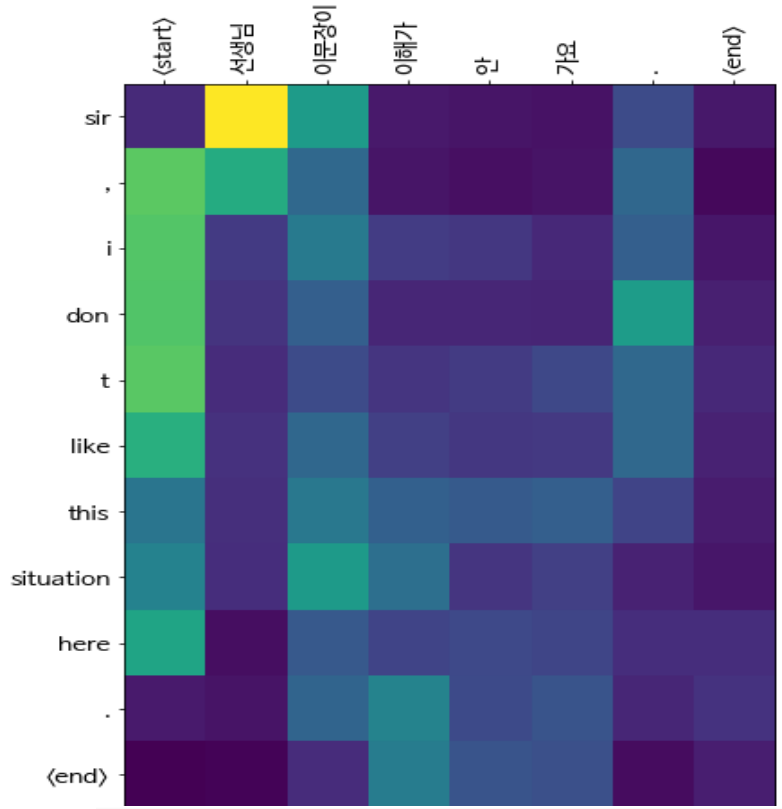
# Keras 모델 구현 - GRU model(1 layer)

***Sequence to Sequence Learning with Neural Networks***

```
Input: <start> 나는 매일 저녁 배트를 만나러 다락방으로 가요 . <end>
Predicted translation: i go to the attic every day to the evening . <end>
```




```
Input: <start> 선생님 이문장이 이해가 안 가요 . <end>
Predicted translation: sir , i don t like this situation here . <end>
```



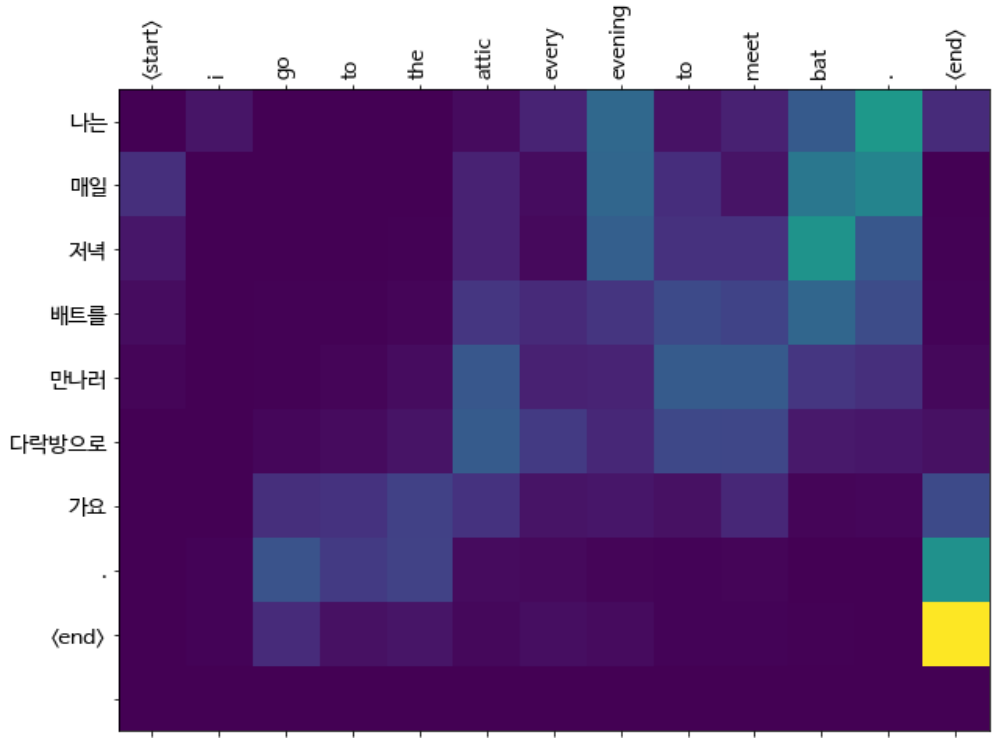

```
[ ]  # right order sentece, 1 layers GRU with 1000 embedding_dims
     bleu_avg(ko[63001:],en[63001:])
```

```
11999/? [04:03<00:00, 49.19it/s]

0.21776383325598686
```

# Keras 모델 구현 - GRU model(1 layer) : EN -> KO

***Sequence to Sequence Learning with Neural Networks***

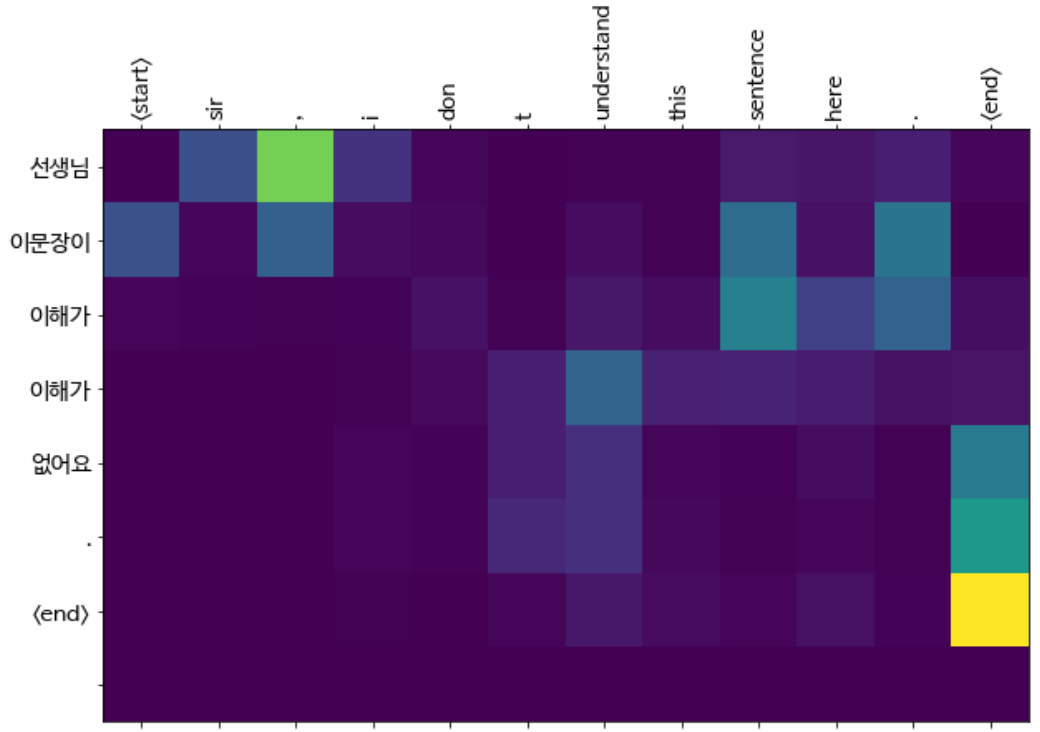
```
Input: <start> i go to the attic every evening to meet bat . <end>
Predicted translation: 나는 매일 저녁 배트를 만나러 다락방으로 가요 . <end>
```




```
[40] # right order sentece, 1 layers GRU with 1000 embedding_dims
     bleu_avg(en[63001:],ko[63001:])
```

```
11999/? [19:41<00:00, 10.15it/s]

0.2021102727536406
```

```
Input: <start> sir , i don t understand this sentence here . <end>
Predicted translation: 선생님 이문장이 이해가 이해가 없어요 . <end>
```



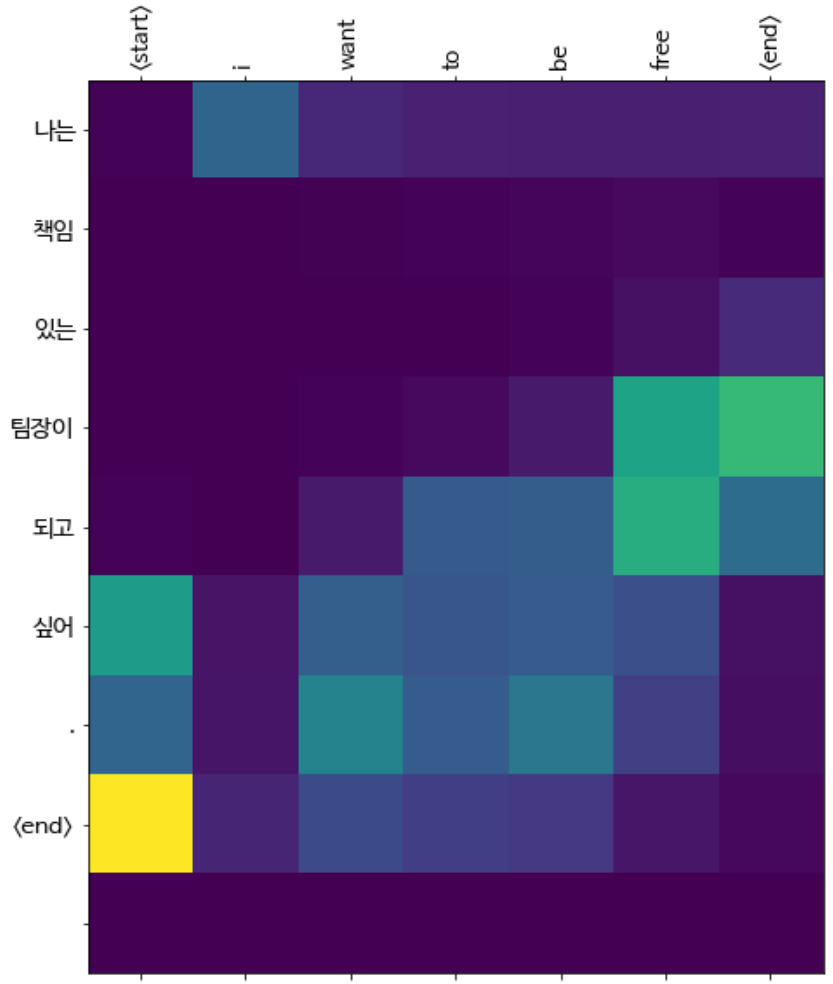

```
[41] # right order sentece, 1 layers GRU with 1000 embedding_dims
     bleu_avg(en[:63000],ko[:63000])
```

```
63000/? [1:52:20<00:00, 9.35it/s]

0.21339550885076813
```

# Keras 모델 구현 - GRU model(1 layer) : EN -> KO

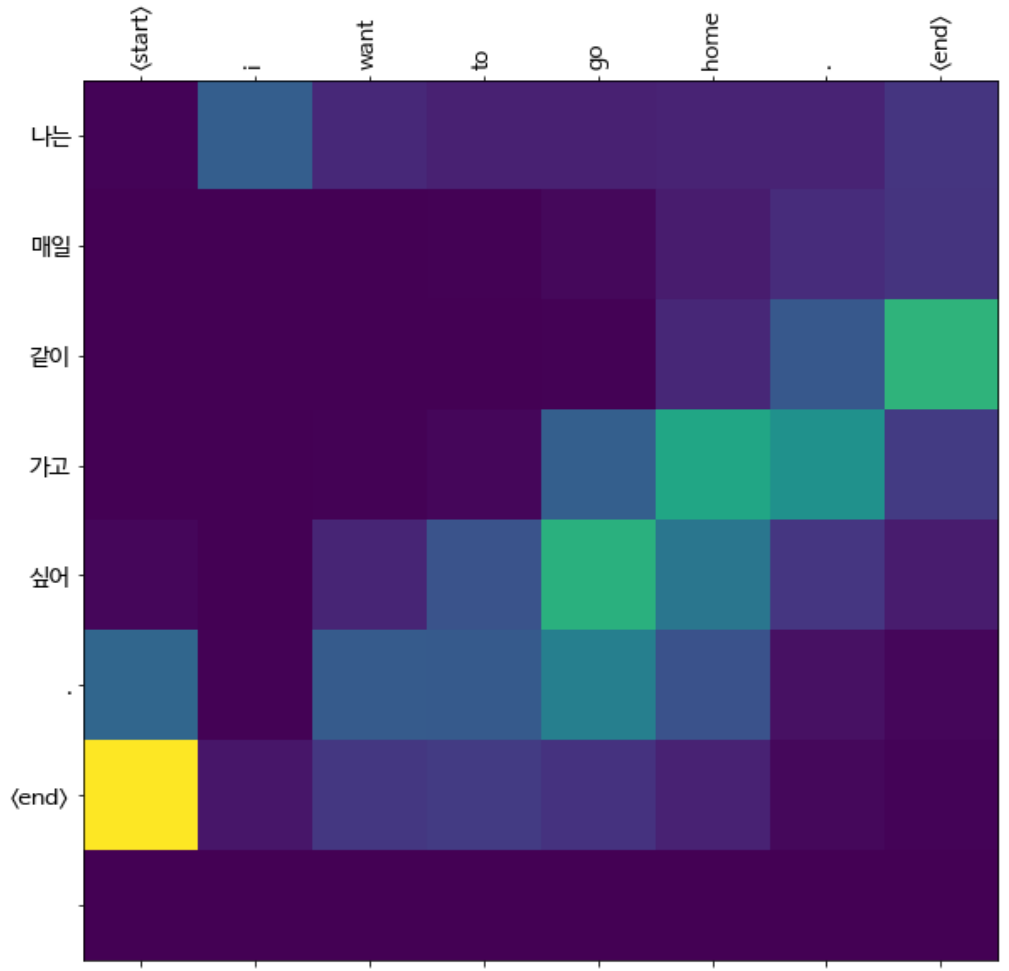
***Sequence to Sequence Learning with Neural Networks***

```
Input: <start> i want to be free <end>
Predicted translation: 나는 책임 있는 팀장이 되고 싶어 . <end>
```




```
translate('I want to go home.')
```

```
Input: <start> i want to go home . <end>
Predicted translation: 나는 매일 같이 가고 싶어 . <end>
```



# Keras 모델 구현 - 결론

*Sequence to Sequence Learning with Neural Networks*

논문 구현을 위해 사용한 LSTM모델보다 Tensorflow에서 제공한 GRU 모델이 성능이 좋음

Train Data에서는 상대적으로 성능이 좋았으나, Test Data에서는 향상된 성능 체감엔 한계가 존재

Embedding dimension의 크기 조정이 LSTM Layer를 쌓는 것보다 성능 향상엔 효과적

Data Set의 크기가 모델의 성능을 좌우

논문과 마찬가지로 역순을 적용한 Train Data를 통해 학습시킨 모델의 성능이 좋았음

한계: Train data에서 Word index가 생성되지 않은 단어에 대해선
Test에서 예측이 불가능

▶ 전처리, Data set의 크기가 중요