

```

1 import tensorflow as tf
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4
5 import matplotlib.ticker as ticker
6 import matplotlib.pyplot as plt
7
8 import time
9 import re
10 import os
11 import io

```

```
1 path_to_zip = tf.keras.utils.get_file('spa-eng.zip', origin='http://storage.goog
```

```

    Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip
2646016/2638744 [=====] - 0s 0us/step
2654208/2638744 [=====] - 0s 0us/step

```

```
1 path_to_file = os.path.dirname(path_to_zip)+"/spa-eng/spa.txt"
```

```

1 with open(path_to_file, "r") as f:
2     raw = f.read().splitlines()
3
4 print("Data Size: ", len(raw))
5 print("Example :")
6
7 for sen in raw[0:100][::20]: print(">>", sen)

```

```

Data Size: 118964
Example :
>> Go. Ve.
>> Wait.      Esperen.
>> Hug me.    Abrázame.
>> No way!    ¡Ni cagando!
>> Call me.   Llamame.

```

```

1 def preprocess_sentence(sentence, s_token=False, e_token=False):
2     # 소문자 변경
3     sentence = sentence.lower().strip()
4
5     # 1. 문장 부호를 \1
6     sentence = re.sub(r"([?!.])", r" \1 ", sentence)
7     # 2. [ ] --> 공백
8     sentence = re.sub(r'[" "]', " ", sentence)
9     # 3. 모든 알파벳, 문장기호를 제외한 것들을 공백으로 바꿔주세요.
10    sentence = re.sub(r"[^a-zA-Z?!.,]+", " ", sentence)
11
12    sentence = sentence.strip()
13
14    if s_token:
15        sentence = '<start> ' + sentence

```

```

16
17     if e_token:
18         sentence += ' <end>'
19
20     return sentence

```

```

1 enc_corpus = []
2 dec_corpus = []
3
4 num_examples = 30000
5
6 for pair in raw[:num_examples]:
7     eng, spa = pair.split("\t")
8
9     enc_corpus.append(preprocess_sentence(eng))
10    dec_corpus.append(preprocess_sentence(spa, s_token=True, e_token=True))
11
12 print("English :", enc_corpus[100])
13 print("Spanish :", dec_corpus[100])

```

```

English : go away !
Spanish : <start> salga de aqu ! <end>

```

```

1 def tokenize(corpus):
2     tokenizer = tf.keras.preprocessing.text.Tokenizer(filters='')
3     tokenizer.fit_on_texts(corpus)
4
5     tensor = tokenizer.texts_to_sequences(corpus)
6     tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor, padding='post')
7
8     return tensor, tokenizer

```

```

1 # 정제된 텍스트를 tokenize() 함수를 사용해 토큰화해서 텐서로 변환하기!
2 enc_tensor, enc_tokenizer = tokenize(enc_corpus)
3 dec_tensor, dec_tokenizer = tokenize(dec_corpus)

```

```

1 # Quiz 1 훈련데이터와 검증데이터를 8:2 분리하세요.
2 enc_train, enc_val, dec_train, dec_val = train_test_split(enc_tensor, dec_tensor)

```

```

1 # Quiz 2
2 # index_word를 활용하여 english vocab size 반환
3 # index_word를 활용하여 spanish vocab size 반환
4
5 print('English Vocab Size :', len(enc_tokenizer.index_word))
6 print('Spanish Vocab Size :', len(dec_tokenizer.index_word))

```

```

English Vocab Size : 4931
Spanish Vocab Size : 8893

```

- Bahdanau Attention

$$Score_{alignment} = W * \tanh(W_{decoder} * H_{decoder} + W_{encoder} * H_{encoder})$$

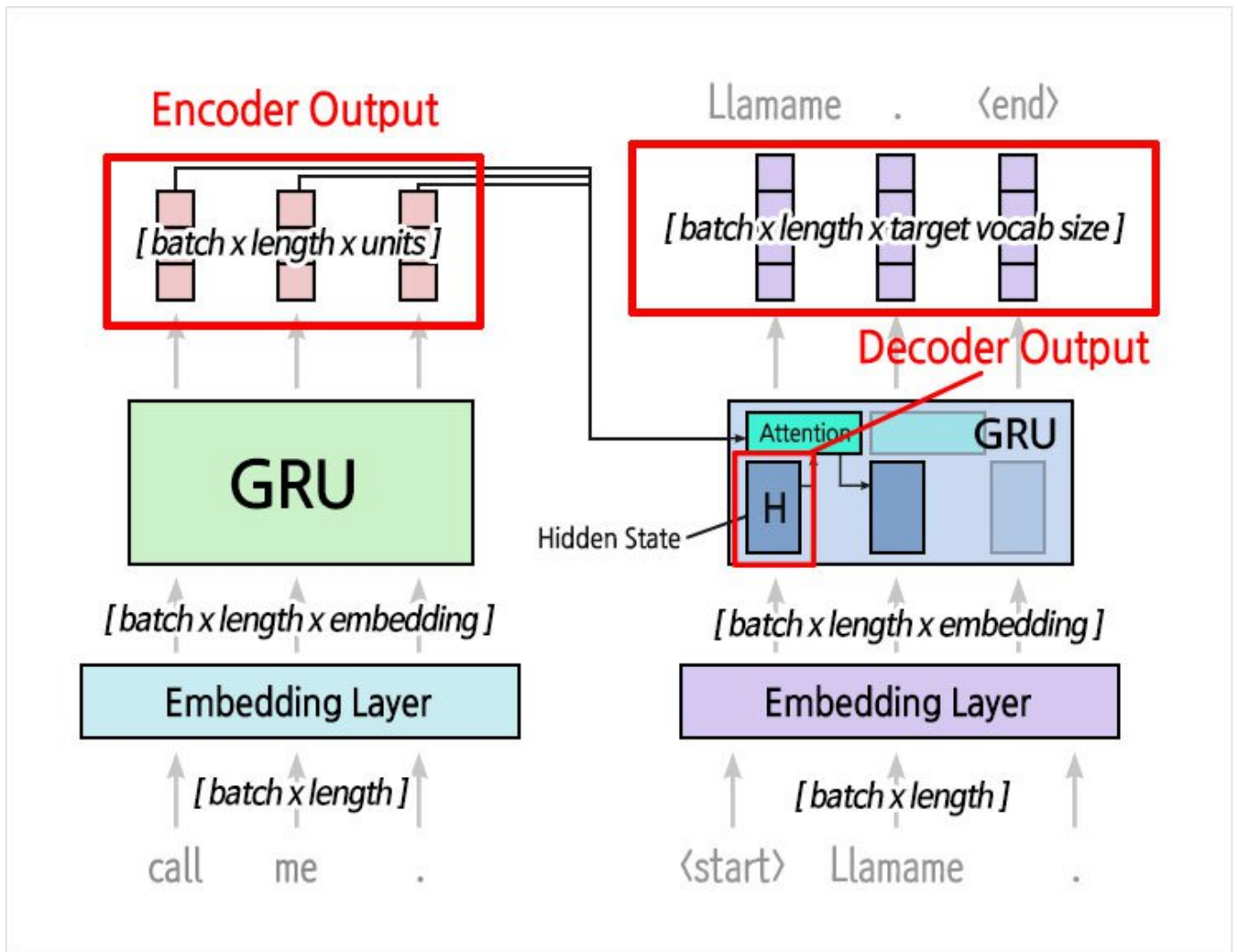
```
1 # 바다나우 어텐션 클래스 만들기
2 class BahdanauAttention(tf.keras.layers.Layer):
3     def __init__(self, units):
4         super(BahdanauAttention, self).__init__()
5         self.w_dec = tf.keras.layers.Dense(units)
6         self.w_enc = tf.keras.layers.Dense(units)
7         self.w_com = tf.keras.layers.Dense(1)
8
9     def call(self, h_enc, h_dec):
10         # h_enc shape : [batch x length x units]
11         # h_dec shape : [batch x units]
12
13         h_enc = self.w_enc(h_enc)
14         h_dec = tf.expand_dims(h_dec, 1)
15         h_dec = self.w_dec(h_dec)
16
17         score = self.w_com(tf.nn.tanh(h_dec + h_enc))
18
19         attn = tf.nn.softmax(score, axis = 1)
20
21         context_vec = attn * h_enc
22         context_vec = tf.reduce_sum(context_vec, axis=1)
23
24         return context_vec, attn

```

```
1 class Encoder(tf.keras.Model):
2     def __init__(self, vocab_size, embedding_dim, enc_units):
3         super(Encoder, self).__init__()
4         # todo
5         self.enc_units = enc_units
6         self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
7         self.gru = tf.keras.layers.GRU(enc_units, return_sequences=True)
8
9     def call(self, x):
10         # todo
11         out = self.embedding(x)
12         out = self.gru(out)
13
14         return out

```



```

1 class Decoder(tf.keras.Model):
2     def __init__(self, vocab_size, embedding_dim, dec_units):
3         super(Decoder, self).__init__()
4         # todo
5         self.dec_units = dec_units
6         self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
7         self.gru = tf.keras.layers.GRU(dec_units, return_sequences=True, return_
8         self.fc = tf.keras.layers.Dense(vocab_size)
9         self.attention = BahdanauAttention(self.dec_units)
10
11     def call(self, x, h_dec, enc_out):
12         # todo
13         context_vec, attn = self.attention(enc_out, h_dec)
14
15         out = self.embedding(x)
16         out = tf.concat([tf.expand_dims(context_vec, 1), out], axis=-1)
17
18         out, h_dec = self.gru(out)
19         out = tf.reshape(out, (-1, out.shape[2]))
20         out = self.fc(out)
21
22         return out, h_dec, attn

```

```

1 BATCH_SIZE = 64
2 src_vocab_size = len(enc_tokenizer.index_word)+1
3 tgt_vocab_size = len(dec_tokenizer.index_word)+1
4
5 units = 1024
6 embedding_dim = 512
7
8 encoder = Encoder(src_vocab_size, embedding_dim, units)
9 decoder = Decoder(tgt_vocab_size, embedding_dim, units)
10
11 # sample input
12 sequence_len = 30
13
14 sample_enc = tf.random.uniform((BATCH_SIZE, sequence_len))
15 sample_output = encoder(sample_enc)
16
17 print('Encoder Output :', sample_output.shape)
18
19 sample_state = tf.random.uniform((BATCH_SIZE, units))
20 sample_logits, h_dec, attn = decoder(tf.random.uniform((BATCH_SIZE, 1)), sample_
21
22 print('Decoder output :', sample_logits.shape)
23 print('Decoder Hidden State :', h_dec.shape)
24 print('Attention :', attn.shape)

```

Encoder Output : (64, 30, 1024)  
Decoder output : (64, 8894)  
Decoder Hidden State : (64, 1024)  
Attention : (64, 30, 1)

## ▼ 훈련하기 1. Optimizer & loss

```

1 optimizer = tf.keras.optimizers.Adam()
2 loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, re
3 # Categorical Crossentropy()
4 # [0.1 0.2 0.7] ----> one hot encoding [0, 0, 1]
5 # SparseCategoricalCrossentropy
6 # [0.1 0.2 0.7] ----> 정수 인덱스 2
7
8 def loss_function(real, pred):
9     mask = tf.math.logical_not(tf.math.equal(real, 0))
10     loss = loss_object(real, pred)
11
12     mask = tf.cast(mask, dtype = loss.dtype)
13     loss *= mask
14
15     return tf.reduce_mean(loss)

```

## ▼ 훈련하기 2.train\_step

## train step 학습과정

1. Encoder에 소스 문장을 전달해 컨텍스트 벡터인 enc\_out을 생성
2. Decoder에 입력으로 전달할 토큰 문장 생성
3. t=0일 때, Decoder의 Hidden state는 Encoder의 Final state로 정의.  $h_{dec} = enc\_out[:, -1]$
4. 문장과 enc\_out, Hidden state를 기반으로 다음단어 (t=1) 예측 pred
5. 예측된 단어와 정답간의 loss를 구한 후, t=1의 정답 단어를 다음 입력으로 사용 (예측단어X)
6. 반복!

```

1 @tf.function
2 def train_step(src, tgt, encoder, decoder, optimizer, dec_tok):
3     bsz = src.shape[0]
4     loss = 0
5
6     with tf.GradientTape() as tape:
7         enc_out = encoder(src)
8         h_dec = enc_out[:, -1]
9
10        dec_src = tf.expand_dims([dec_tok.word_index['<start>']] * bsz, 1)
11
12        for t in range(1, tgt.shape[1]):
13            pred, h_dec, _ = decoder(dec_src, h_dec, enc_out)
14            loss += loss_function(tgt[:, t], pred)
15            dec_src = tf.expand_dims(tgt[:, t], 1)
16
17        batch_loss = (loss / int(tgt.shape[1]))
18
19        variables = encoder.trainable_variables + decoder.trainable_variables
20        gradients = tape.gradient(loss, variables)
21        optimizer.apply_gradients(zip(gradients, variables))
22
23    return batch_loss

```

```
1 !pip install tqdm
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages
```

```

1 from tqdm import tqdm
2 import random
3
4 epochs = 20
5
6 for epoch in range(epochs):
7     total_loss = 0
8     idx_list = list(range(0, enc_train.shape[0], BATCH_SIZE))
9     random.shuffle(idx_list)
10    t = tqdm(idx_list)
11
12    for (batch, idx) in enumerate(t):

```

```

13         batch_loss = train_step(enc_train[idx:idx+BATCH_SIZE],
14                                 dec_train[idx:idx+BATCH_SIZE],
15                                 encoder,
16                                 decoder,
17                                 optimizer,
18                                 dec_tokenizer)
19         total_loss += batch_loss
20
21 t.set_description_str('Epoch %2d' % (epoch+1))
22 t.set_postfix_str('Loss %.4f' % (total_loss.numpy()/(batch+1)))

```

```

100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.65it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]

```

## ▼ Evaluate step

```

1 @tf.function
2 def eval_step(src, tgt, encoder, decoder, dec_tok):
3     bsz = src.shape[0]
4     loss = 0
5
6     enc_out = encoder(src)
7     h_dec = enc_out[:, -1]
8
9     dec_src = tf.expand_dims([dec_tok.word_index['<start>']] * bsz, 1)
10
11     for t in range(1, tgt.shape[1]):
12         pred, h_dec, _ = decoder(dec_src, h_dec, enc_out)
13         loss += loss_function(tgt[:, t], pred)
14         dec_src = tf.expand_dims(tgt[:, t], 1)
15
16     batch_loss = (loss/int(tgt.shape[1]))
17
18     return batch_loss

```

1 # Training Process

```

2 from tqdm import tqdm
3
4 EPOCHS = 10
5
6 for epoch in range(EPOCHS):
7     total_loss = 0
8
9     idx_list = list(range(0, enc_train.shape[0], BATCH_SIZE))
10    random.shuffle(idx_list)
11    t = tqdm(idx_list)
12
13    for (batch, idx) in enumerate(t):
14        batch_loss = train_step(enc_train[idx:idx+BATCH_SIZE],
15                                dec_train[idx:idx+BATCH_SIZE],
16                                encoder,
17                                decoder,
18                                optimizer,
19                                dec_tokenizer)
20        total_loss += batch_loss
21
22    t.set_description_str('Epoch %2d' % (epoch + 1))
23    t.set_postfix_str('Loss %.4f' % (total_loss.numpy() / (batch + 1)))
24
25    test_loss = 0
26
27    idx_list = list(range(0, enc_val.shape[0], BATCH_SIZE))
28    random.shuffle(idx_list)
29    t = tqdm(idx_list)
30
31    for (test_batch, idx) in enumerate(t):
32        test_batch_loss = eval_step(enc_val[idx:idx+BATCH_SIZE],
33                                    dec_val[idx:idx+BATCH_SIZE],
34                                    encoder,
35                                    decoder,
36                                    dec_tokenizer)
37        test_loss += test_batch_loss
38
39    t.set_description_str('Test Epoch %2d' % (epoch + 1))
40    t.set_postfix_str('Test Loss %.4f' % (test_loss.numpy() / (test_batch + 1)))

```

```

100%|██████████████████| 375/375 [01:42<00:00, 3.64it/s]
100%|██████████████████| 94/94 [00:24<00:00, 3.77it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.65it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.23it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.65it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.23it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.64it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.23it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.64it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.25it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.64it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.23it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.65it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.20it/s]
100%|██████████████████| 375/375 [01:42<00:00, 3.65it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.24it/s]

```



```

100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.24it/s]
100%|██████████████████| 375/375 [01:43<00:00, 3.63it/s]
100%|██████████████████| 94/94 [00:09<00:00, 10.24it/s]

```

```

1 def evaluate(sentence, encoder, decoder):
2     attention = np.zeros((dec_train.shape[-1], enc_train.shape[-1]))
3     sentence = preprocess_sentence(sentence)
4     inputs = enc_tokenizer.texts_to_sequences([sentence.split()])
5     inputs = tf.keras.preprocessing.sequence.pad_sequences(inputs,
6                                                             maxlen = enc_train.shap
7                                                             padding= 'post')
8
9     result = ''
10    enc_out = encoder(inputs)
11    dec_hidden = enc_out[:, -1]
12    dec_input = tf.expand_dims([dec_tokenizer.word_index['<start>']], 0)
13
14    for t in range(dec_train.shape[-1]):
15        predictions, dec_hidden, attention_weights = decoder(dec_input,
16                                                            dec_hidden,
17                                                            enc_out)
18
19        attention_weights = tf.reshape(attention_weights, (-1, ))
20        attention[t] = attention_weights.numpy()
21
22        predicted_id = \
23            tf.argmax(tf.math.softmax(predictions, axis=-1)[0]).numpy()
24
25        result += dec_tokenizer.index_word[predicted_id] + ' '
26
27        if dec_tokenizer.index_word[predicted_id] == '<end>':
28            return result, sentence, attention
29
30        dec_input = tf.expand_dims([predicted_id], 0)
31
32    return result, sentence, attention
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

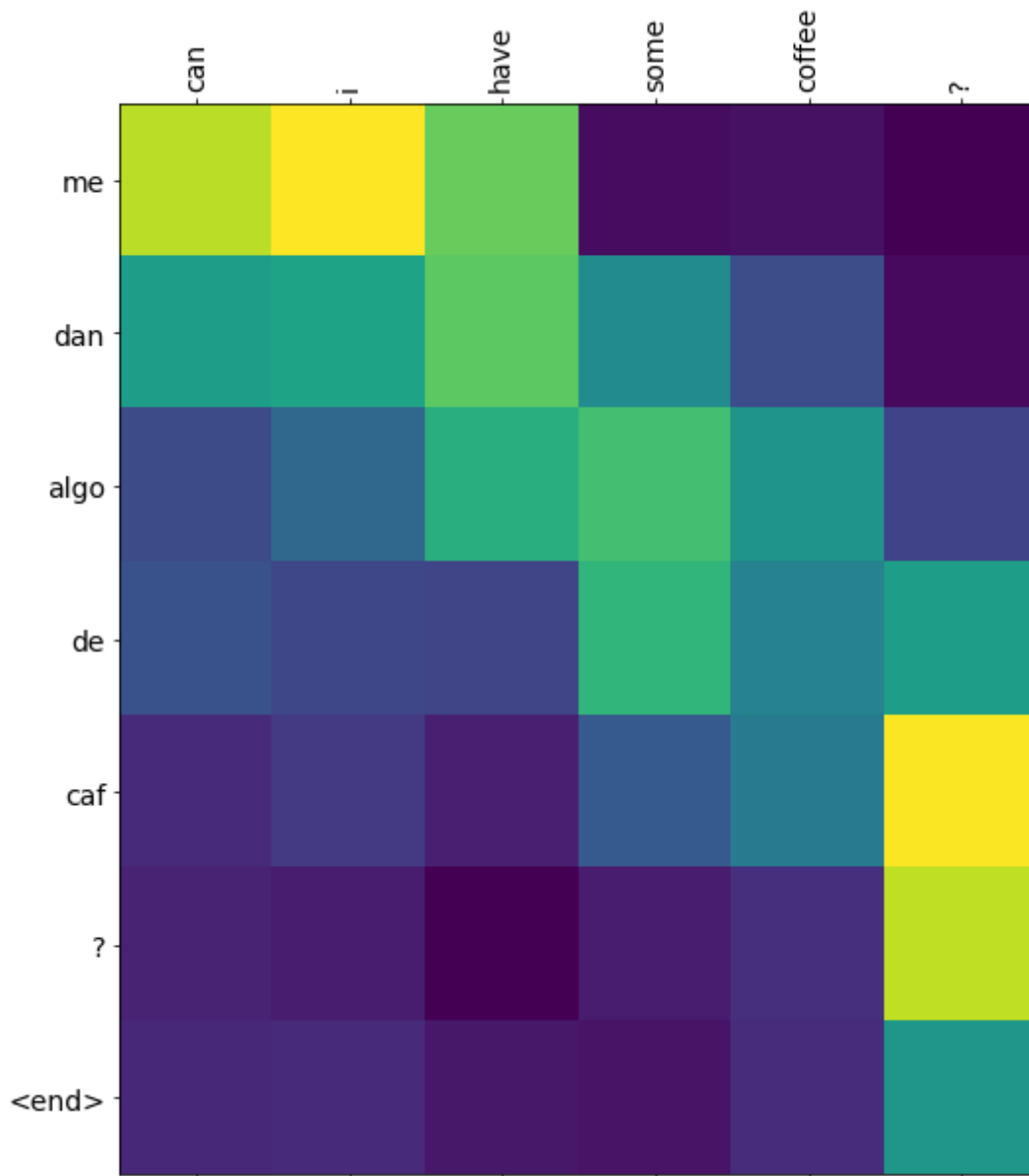
1 def translate(sentence, encoder, decoder):
2     result, sentence, attention = evaluate(sentence, encoder, decoder)
3
4     print('Input : %s' % (sentence))
5     print('Predicted translation : {}'.format(result))
6
7     attention = attention[:len(result.split()), :len(sentence.split())]
8     plot_attention(attention, sentence.split(), result.split(' '))

```

```
1 translate("Can I have some coffee?", encoder, decoder)
```

Input : can i have some coffee ?

Predicted translation : me dan algo de caf ? <end>



## ▼ Transformer

기존의 seq2seq 모델의 한계

- 입력 시퀀스를 하나의 벡터표현으로 압축 (context vector) 디코더는 이를 통해 출력 시퀀스를 만들어냄
- 정보가 일부 손실된다는 단점

- $d_{model} = 512$
- num\_layers = 6
- num\_heads = 8
- $d_{ff} = 2048$

## ▼ 포지셔널 인코딩

$$PE_{(pose, 2i)} = \sin(pos/1000^{2i/d_{model}})$$

$$PE_{(pose, 2i + 1)} = \cos(pos/1000^{2i/d_{model}})$$

```

1 import tensorflow as tf
2 import matplotlib.pyplot as plt
3 import numpy as np

1 class PositionalEncoding(tf.keras.layers.Layer):
2     def __init__(self, position, d_model):
3         super(PositionalEncoding, self).__init__()
4         self.pos_encoding = self.positional_encoding(position, d_model)
5
6     def get_angles(self, position, i, d_model):
7         angles = 1 / tf.pow(10000, (2 * (i // 2)) / tf.cast(d_model, tf.float32))
8         return position * angles
9
10    def positional_encoding(self, position, d_model):
11        angle_rads = self.get_angles(
12            position=tf.range(position, dtype=tf.float32)[: , tf.newaxis],
13            i=tf.range(d_model, dtype=tf.float32)[tf.newaxis, :],
14            d_model=d_model)
15
16        # 배열의 짝수 인덱스(2i)에는 사인 함수 적용
17        sines = tf.math.sin(angle_rads[:, 0::2])
18
19        # 배열의 홀수 인덱스(2i+1)에는 코사인 함수 적용
20        cosines = tf.math.cos(angle_rads[:, 1::2])
21
22        angle_rads = np.zeros(angle_rads.shape)
23        angle_rads[:, 0::2] = sines
24        angle_rads[:, 1::2] = cosines
25        pos_encoding = tf.constant(angle_rads)
26        pos_encoding = pos_encoding[tf.newaxis, ...]
27
28        print(pos_encoding.shape)
29        return tf.cast(pos_encoding, tf.float32)
30
31    def call(self, inputs):
32        return inputs + self.pos_encoding[:, :tf.shape(inputs)[-1], :]

```

```

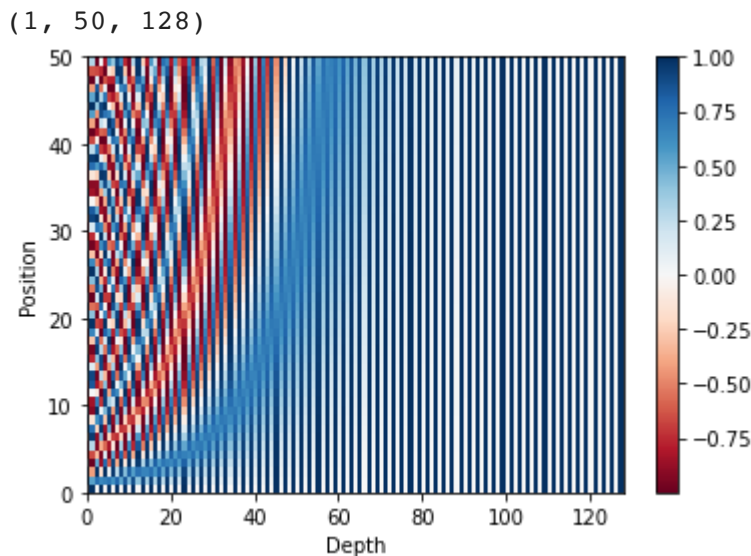
32 return inputs + self.pos_encoding[:, :inputs.shape[-1], :]

```

```

1 # 50x 128크기를 가지는 포지셔널 인코딩 행렬을 시각화하기
2 # 입력문자의 단어가 50 각 단어가 128차원의 임베딩 벡터를 가질 때 사용하는 행렬
3
4 sample_pos_encoding = PositionalEncoding(50, 128)
5
6 plt.pcolormesh(sample_pos_encoding.pos_encoding.numpy()[0], cmap = "RdBu")
7 plt.xlabel('Depth')
8 plt.xlim((0, 128))
9 plt.ylabel('Position')
10 plt.colorbar()
11 plt.show()

```



```

1 def scaled_dot_product_attention(query, key, value, mask):
2     # query 크기 : (batch_size, num_heads, query의 문장 길이, d_model/num_heads)
3     # key 크기 : (batch_size, num_heads, key의 문장 길이, d_model/num_heads)
4     # value 크기 : (batch_size, num_heads, value의 문장 길이, d_model/num_heads)
5     # padding mask : (batch_size, 1, 1, key의 문장 길이)
6
7     # Q와 K의 곱
8     matmul_qk = tf.matmul(query, key, transpose_b=True)
9
10    # 스케일링 : dk의 루트값으로 나눈다.
11    depth = tf.cast(tf.shape(key)[-1], tf.float32)
12    logits = matmul_qk / tf.math.sqrt(depth)
13
14    # 마스킹 어텐션 스코어 행렬의 마스킹 할 위치에 매우 작은 음수값을 넣는다.
15    # 매우 작은 값이므로 소프트맥스 함수를 지나면 행렬의 해당 위치의 값은 0이 된다.
16    if mask is not None:
17        logits += (mask * -1e9)
18
19    # 소프트맥스 함수는 마지막 차원인 key의 문장 길이 방향으로 수행된다.
20    # attention weight : (batch_size, num_heads, query의 문장 길이, key의 문장 길이)
21    attention_weight = tf.nn.softmax(logits, axis=-1)
22
23    # output : (batch_size, num_heads, query의 문장 길이 , d_model/num_heads)
24    output = tf.matmul(attention_weight, value)

```

25

26 return output, attention\_weight

1 # scaled\_dot\_product\_attention함수가 정상적으로 작동하는지 테스트 !

2 np.set\_printoptions(suppress=True)

3 temp\_k = tf.constant([[10, 0, 0],

4 [0, 10, 0],

5 [0, 0, 10],

6 [0, 0, 10]], dtype=tf.float32) # (4, 3)

7 temp\_v = tf.constant([[ 1, 0],

8 [ 10, 0],

9 [ 100, 5],

10 [1000, 6]], dtype=tf.float32) # (4, 2)

11 temp\_q = tf.constant([[0, 10, 0]], dtype = tf.float32) # (1, 3)

1 temp\_out, temp\_attn = scaled\_dot\_product\_attention(temp\_q, temp\_k, temp\_v, None)

2 print(temp\_attn) # 어텐션 분포

3 print(temp\_out) # 어텐션 값

tf.Tensor([[0. 1. 0. 0.]], shape=(1, 4), dtype=float32)

tf.Tensor([[10. 0.]], shape=(1, 2), dtype=float32)

1 # scaled\_dot\_product\_attention함수가 정상적으로 작동하는지 테스트 !

2 np.set\_printoptions(suppress=True)

3 temp\_k = tf.constant([[10, 0, 0],

4 [0, 10, 0],

5 [0, 0, 10],

6 [0, 0, 10]], dtype=tf.float32) # (4, 3)

7 temp\_v = tf.constant([[ 1, 0],

8 [ 10, 0],

9 [ 100, 5],

10 [1000, 6]], dtype=tf.float32) # (4, 2)

11 temp\_q = tf.constant([[0, 0, 10]], dtype = tf.float32) # (1, 3)

1 temp\_out, temp\_attn = scaled\_dot\_product\_attention(temp\_q, temp\_k, temp\_v, None)

2 print(temp\_attn) # 어텐션 분포

3 print(temp\_out) # 어텐션 값

tf.Tensor([[0. 0. 0.5 0.5]], shape=(1, 4), dtype=float32)

tf.Tensor([[550. 5.5]], shape=(1, 2), dtype=float32)

1 temp\_k = tf.constant([[10, 0, 0],

2 [0, 10, 0],

3 [0, 0, 10],

4 [0, 0, 10]], dtype=tf.float32) # (4, 3)

5 temp\_v = tf.constant([[ 1, 0],

6 [ 10, 0],

7 [ 100, 5],

8 [1000, 6]], dtype=tf.float32) # (4, 2)

9 temp\_q = tf.constant([[0, 0, 10],

```

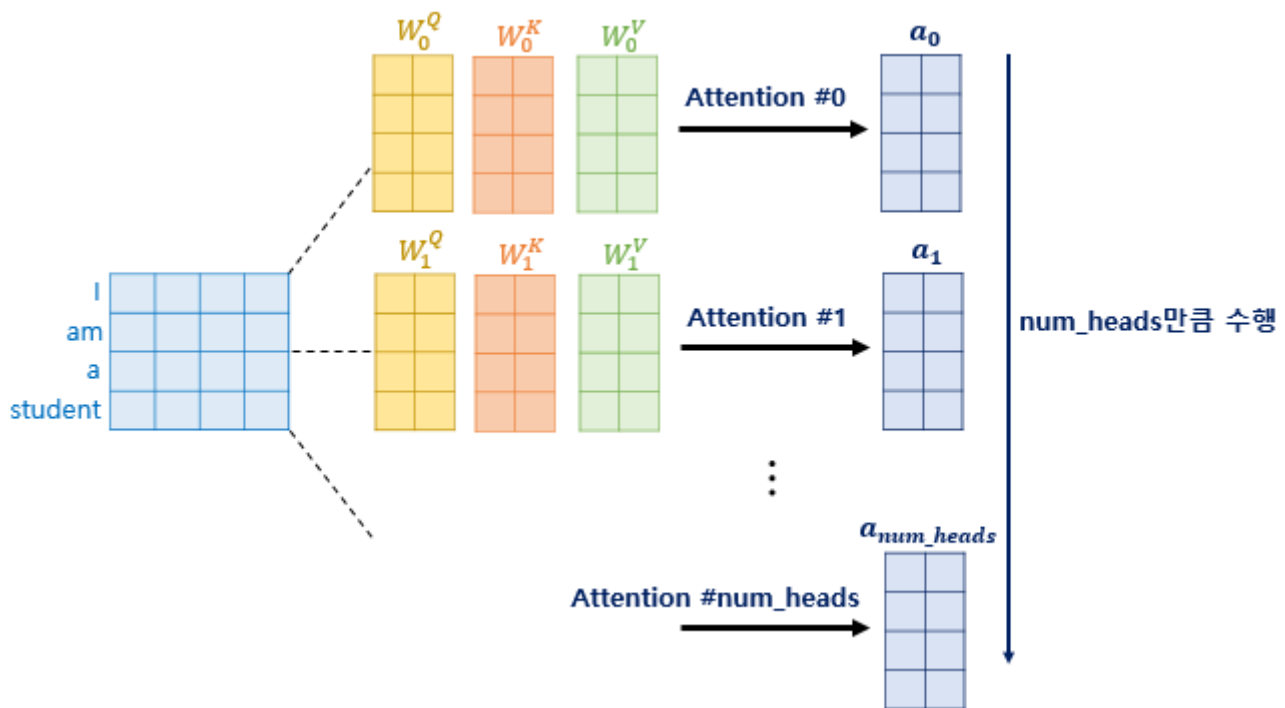
10         [0, 10, 0],
11         [10, 10, 0]], dtype = tf.float32) # (1, 3)

1 temp_out, temp_attn = scaled_dot_product_attention(temp_q, temp_k, temp_v, None)
2 print(temp_attn) # 어텐션 분포
3 print(temp_out) # 어텐션 값

tf.Tensor(
[[0.  0.  0.5 0.5]
 [0.  1.  0.  0. ]
 [0.5 0.5 0.  0. ]], shape=(3, 4), dtype=float32)
tf.Tensor(
[[550.    5.5]
 [ 10.     0. ]
 [  5.5    0. ]], shape=(3, 2), dtype=float32)

```

## ▼ 멀티헤드 어텐션



병렬 어텐션의 효과?

머리가 여러개이기 때문에, 여러 시점에서 상대방을 볼 수있다.

1.  $W_Q, W_K, W_V$ 에 해당하는  $d_{model}$  크기의 밀집층(Dense layer)을 지남
2. 지정된 헤드 수 (num\_heads)만큼 나눈다.
3. 스케일드 닷 프로덕트 어텐션
4. 나눠졌던 헤드들을 연결한다.
5.  $W_O$ 에 해당하는 밀집층을 지나게 된다.

```

1 class MultiHeadAttention(tf.keras.layers.Layer):
2     def __init__(self, d_model, num_heads, name="multi_head_attention"):

```

```

3      super(MultiHeadAttention, self).__init__(name=name)
4      self.num_heads = num_heads
5      self.d_model = d_model
6
7      assert d_model % self.num_heads == 0
8
9      self.depth = d_model // self.num_heads
10     self.query_dense = tf.keras.layers.Dense(units=d_model)
11     self.key_dense = tf.keras.layers.Dense(units=d_model)
12     self.value_dense = tf.keras.layers.Dense(units=d_model)
13
14     # wQ에 해당하는 dense
15     self.dense = tf.keras.layers.Dense(units=d_model)
16
17     # num_heads 갯수만 큼 q, k,v를 split하는 함수
18
19     def split_heads(self, inputs, batch_size):
20         inputs = tf.reshape(
21             inputs, shape=(batch_size, -1, self.num_heads, self.depth)
22         )
23         return tf.transpose(inputs, perm=[0,2,1,3])
24
25     def call(self, inputs):
26         query, key, value, mask = inputs['query'], inputs['key'], inputs['value']
27         batch_size = tf.shape(query)[0]
28         # 1. wq, wk, wv에 해당하는 밀집층 지나기
29         # q : (batch_size, query의 문장 길이, d_model)
30         # k : (batch_size, key의 문장 길이, d_model)
31         # v : (batch_size, value의 문장 길이, d_model)
32
33         # 참고 ** 인코더 (k, v)- 디코더(q) 어텐션에서는 query길이와 key, value의 길이는 다를
34         query = self.query_dense(query)
35         key = self.key_dense(key)
36         value = self.value_dense(value)
37
38         # 2. 헤드 나누기
39         # q : (batch_size, num_heads, query의 문장 길이, d_model/num_heads)
40         # k : (batch_size, num_heads, key의 문장 길이, d_model/num_heads)
41         # v : (batch_size, num_heads, value의 문장 길이, d_model/num_heads)
42         query = self.split_heads(query, batch_size)
43         key = self.split_heads(key, batch_size)
44         value = self.split_heads(value, batch_size)
45
46         # 3. 스케일 닷 프로덕트 어텐션
47         # (batch_size, num_heads, query의 문장 길이, d_model/num_heads)
48         scaled_attention, _ = scaled_dot_product_attention(query, key, value, ma
49         # (batch_size, query의 문장 길이, num_heads, d_model/num_heads)
50         scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])
51
52         # 4. 헤드 연결
53         # (batch_size, query의 문장 길이, d_model)
54         concat_attention = tf.reshape(scaled_dot_product_attention, (batch_size,
55
56         # 5. wo에 해당하는 dence층 지나기
57         # (batch_size, query의 문장 길이, d_model)

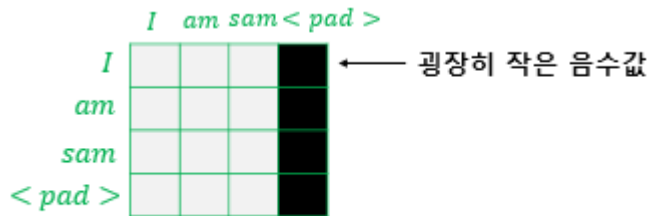
```

```

58         outputs = self.dense(concat_attention)
59
60         return outputs
61

```

## ▼ 패딩 마스크



**Attention Score Matrix**

```

1 def create_padding_mask(x):
2     mask = tf.cast(tf.math.equal(x, 0), tf.float32)
3     # (batch_size, 1, 1, key의 문장 길이)
4     return mask[:, tf.newaxis, tf.newaxis, :]

1 print(create_padding_mask(tf.constant([[1, 21, 777, 0, 0]])))

tf.Tensor([[[[0. 0. 0. 1. 1.]]]], shape=(1, 1, 1, 5), dtype=float32)

```

## ▼ 포지션 와이드 피드 포워드 신경망

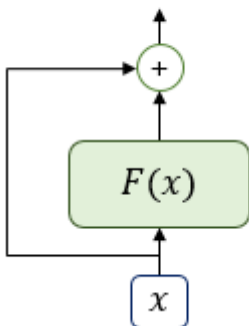
```

outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention)
outpus = tf.kears.layers.Dense(units=d_model)(outpus)

```

## ▼ 잔차 연결과 층 정규화

$$H(x) = x + F(x)$$





## ▼ 인코더 구현하기

```

1 def encoder_layer(dff, d_model, num_heads, dropout, name="encoder_layer"):
2     inputs = tf.keras.Input(shape=(None, d_model), name="inputs")
3
4     # 인코더는 패딩 마스크 사용
5     padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")
6
7     # 멀티 헤드 어텐션
8     attention = MultiHeadAttention(
9         d_model, num_heads, name='attention')({
10         'query':inputs, 'key':inputs, 'value': inputs,
11         'mask' : padding_mask #패딩 마스크 사용
12     })
13
14     # 드롭아웃 + 잔차 연결과 층 정규화
15     attention = tf.keras.layers.Dropout(rate=dropout)(attention)
16     attention = tf.keras.layers.LayerNormalization(epsilon=1e-6)(inputs + attent
17
18     #포지션 와이즈 피드 포워드 신경망 (두번째 서브층)
19     outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention)
20     outputs = tf.keras.layers.Dense(units=d_model)(outputs)
21
22     # 드롭아웃 + 잔차 연결 과 층 정규화
23     outputs = tf.keras.layers.Dropout(rate=dropout)(outputs)
24     outputs = tf.keras.layers.LayerNormalization(epsilon=1e-6)(attention + outpu
25
26     return tf.keras.Model(inputs=[inputs, padding_mask], outputs = outputs, name

```

## ▼ 인코더 쌓기

```

1 def encoder(vocab_size, num_layers, dff, d_model, num_heads, dropout, name='enco
2     inputs = tf.keras.Input(shape=(None, ), name="inputs")
3
4     #인코더는 패딩마스크 사용
5     padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")
6
7     #포지셔널 인코딩 + 드롭아웃
8     embeddings = tf.keras.layers.Embedding(vocab_size, d_model)(inputs)
9     embeddings *= tf.math.sqrt(tf.cast(d_model, tf.float32))
10    embeddings = PositionalEncoding(vocab_size, d_model)
11    outputs = tf.keras.layers.Dropout(rate=dropout)(embeddings)
12
13    # 인코딩을 num_layer개 쌓기
14    for i in range(num_layers):
15        outputs = encoder_layer(dff=dff, d_model=d_model, num_heads=num_heads, d
16    return tf.keras.Model(inputs=[inputs, padding_mask], outputs = outputs, name

```

