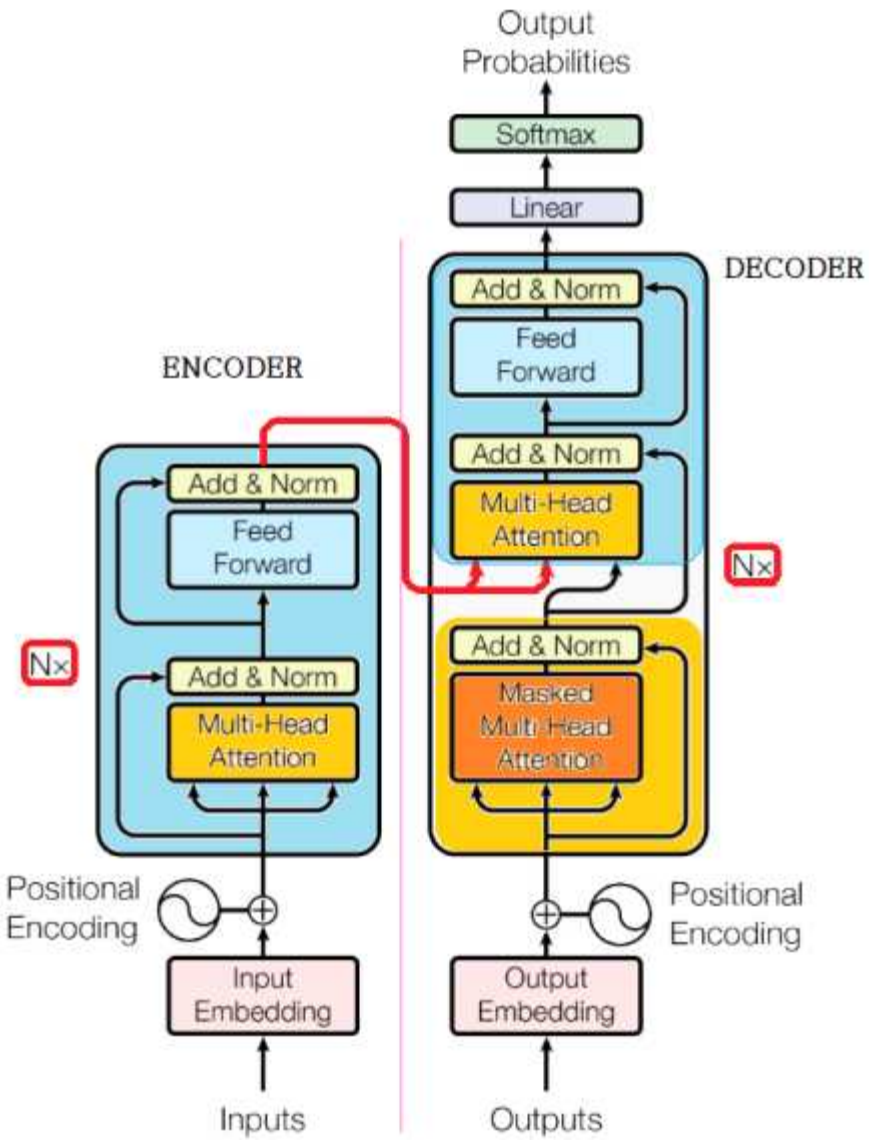
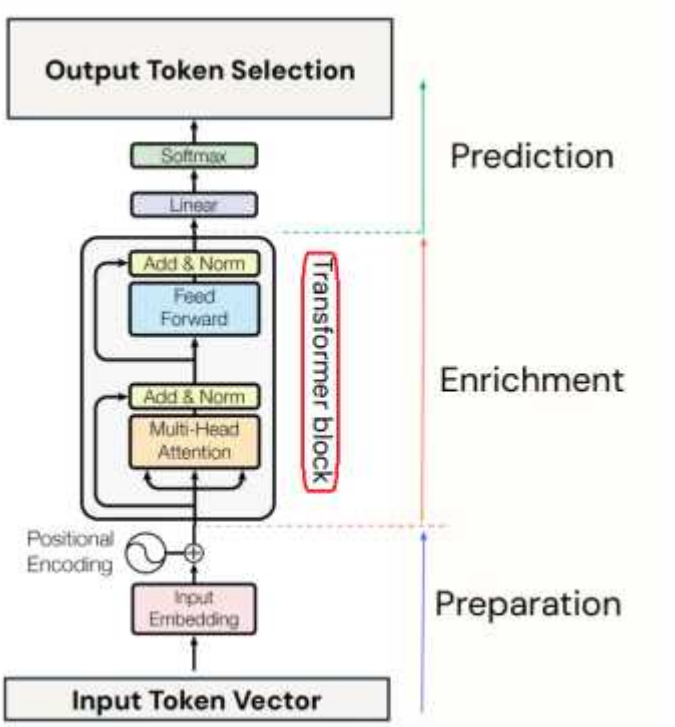
# Transformer

## 1. Transformer Model

- Attention 기반의 Encoder-Decoder 모델
- Positional Encoding의 추가로 입력 벡터의 위치 정보 추가
- Attention을 이용하여 병렬처리 가능하게 한 모델
- Decoder부분에 Masked Multi-Head Attention 적용을 미래 벡터 미반영
- Feed-forward Layer 추가로 비선형성 추가
- Residual Layer와 Layer Normalization 추가로 학습 속도와 학습 안정성 향상
- Encoder 부분을 이용한 BERT 모델과 DECODER 부분을 이용한 GPT 모델의 원조



◉ Sequence Input, Transformer Block, Prediction 부분의 구성




◉ (1) Preparation Phase :

- Word Token -> Word Embedding
- Positional Encoding : input sequence에 순서(order)정보 추가

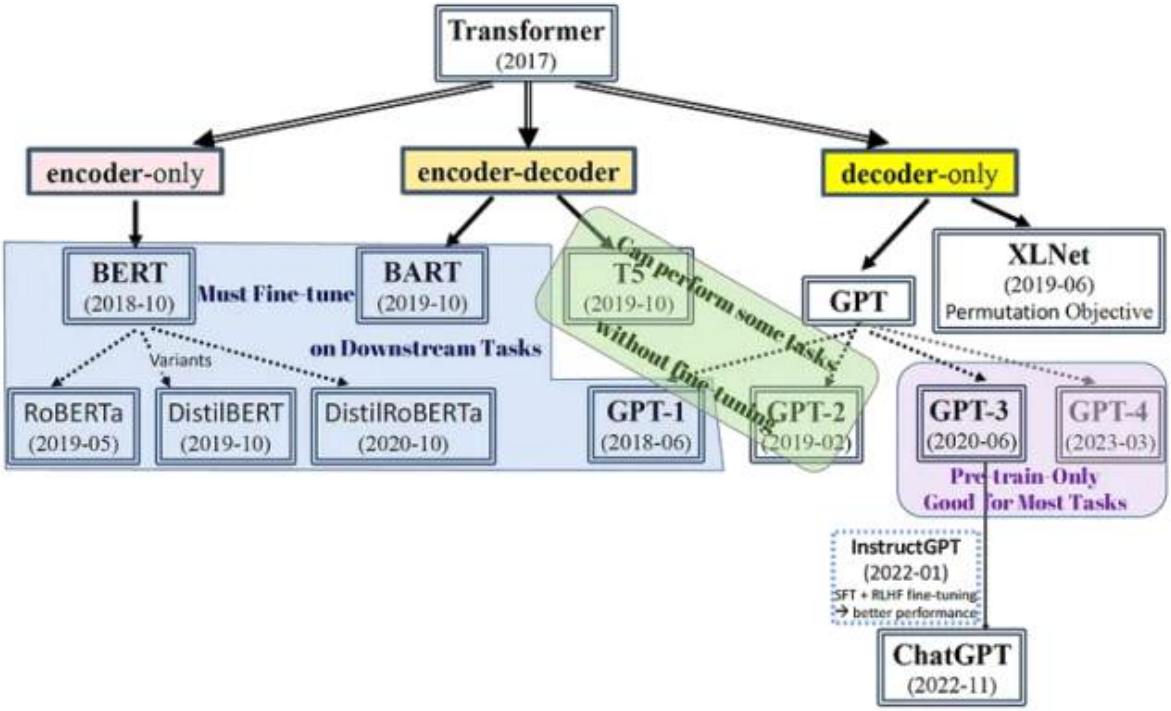
◉ (2) Enrichment Phase :

- Self-Attention : 문장 내 단어 간의 관계성을 표현
- Multi-head (Attention) :
  . Early Attention : 이웃한 입력 벡터간의 근거리 의존성을 표현
  . Middle Attention : 보다 먼 입력 벡터간의 의미(meanings)를 표현
  . Late Attention : 고차원의 요약(추상화)을 표현
- Position-wise Feed Forward NN : attention후 표현된 벡터들에 대한 비선형성 추가
- Residual Connection : Vanishing gradient 문제를 해결
- Layer Normalization : 입력 분포를 균일하게 하여 학습과정을 안정화

◉ (3) Prediction Phase :

- Transformer Block의 출력인 context-aware output vector를 vocabulary_size의 dimension으로 projection 시키고, softmax 함수로 vocabulary들에 대한 확률값 분포를 구함

◉ Transformer를 기반으로 한 3가지 유형의 LLM




(https://medium.com/@akriti.upadhyay/building-custom-gpt-with-pytorch-59e5ba8102d4)

◉ Transformer, Bert의 구조

| | Transformer | Bert_Base | Bert_Large |
|---|---|---|---|
| Multi Heads # | 8 | 12 | 16 |
| Embedding Dimension | 512 | 768 | 1024 |
| Max. Token Size | 512 | 512 | 512 |
| repeated layers # | 6 | 12 | 24 |
| Parameters # | 62 M(Base) / 214 M(Large) | 110 M | 340 M |

◉ **Encoder-only Transformers** : understanding and encoding input sequences
- text classification, named entity recognition, and sentiment analysis
- BERT 계열

◉ **Decoder-only Transformers** : generating output sequences from input sequences
- text generation, machine translation, and summarization
- GPT 계열

◉ **Encoder-Decoder Transformers(Cross-Attention)** : original transformer
- language translation and text summarization
- T5, Bart, Pegasus, ProphetNet, Marge, etc.

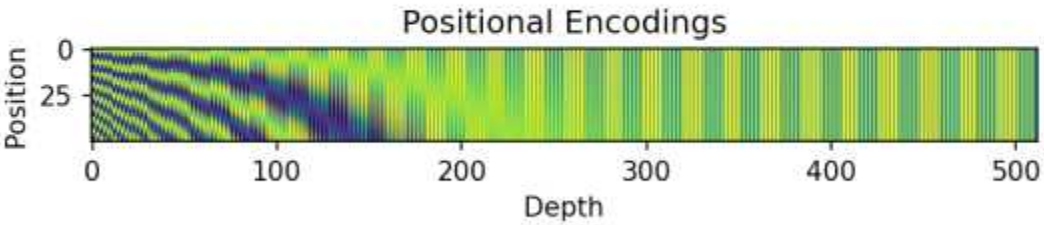
## 2. pytorch를 이용한 Transformer_decoder Model 학습

### 2.1 Positional Encoding Source Code

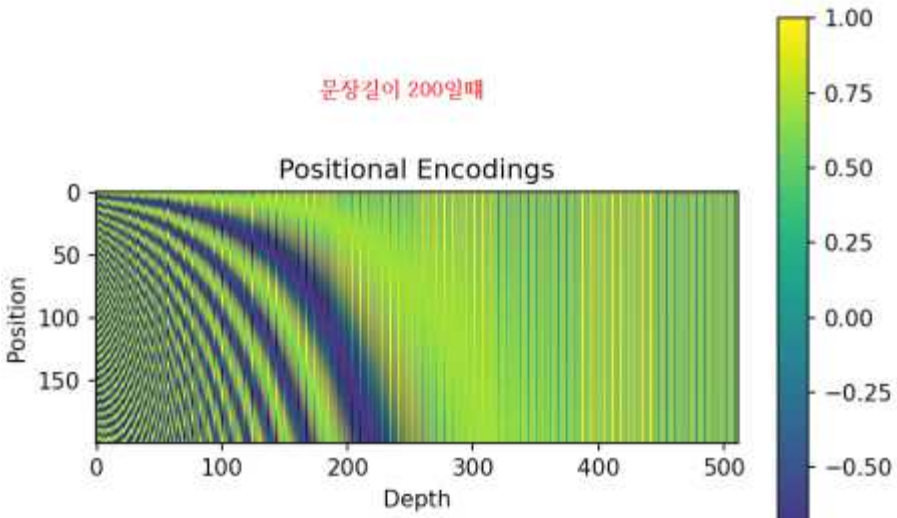
```
import numpy as np
import matplotlib.pyplot as plt
#============================================================
# """Positional Encoding 생성 함수"""
def get_positional_encoding(max_len,d_model):

    pos_enc =np.array([
        [pos /np.power(10000,2 *(i//2)/d_model)for i in range(d_model)]
        for pos in range(max_len)])

    pos_enc[:,0::2]=np.sin(pos_enc[:,0::2])  # 짝수 인덱스에 대한 값 저장
    pos_enc[:,1::2]=np.cos(pos_enc[:,1::2])  # 홀수 인덱스 "

    return pos_enc
max_len =16   # 문장의 최대 길이
d_model =512  # 임베딩 차원
positional_encodings =get_positional_encoding(max_len,d_model)
plt.figure()
plt.imshow(positional_encodings,interpolation='none',cmap='viridis')
plt.colorbar()
plt.title('Positional Encodings')
plt.xlabel('Depth')
plt.ylabel('Position')
plt.grid(visible=False)
plt.show()
```

※ 문장 길이별 positional encoding 값 비교(next page)

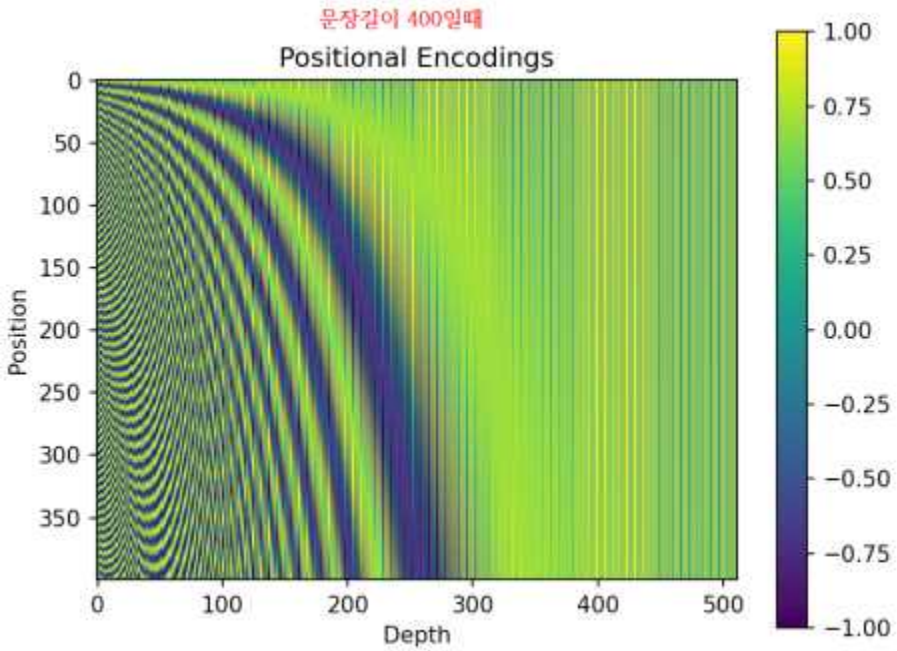
문장길이 50 일때



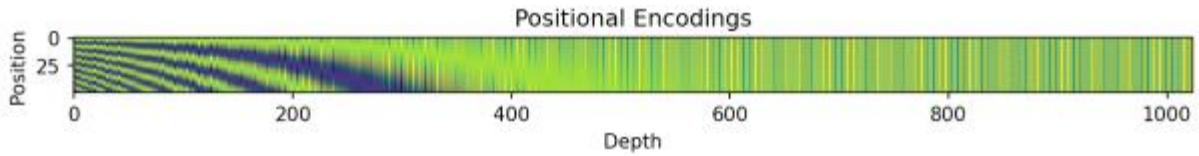

문장길이 200일때



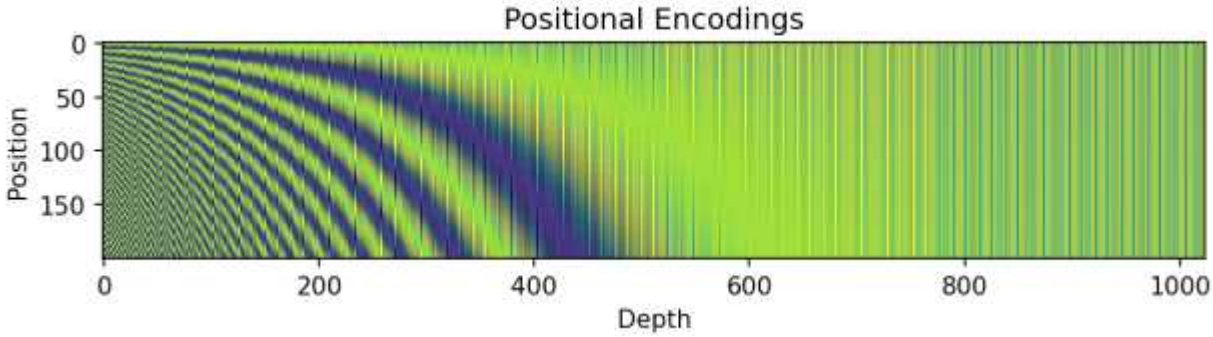

문장길이 400일때



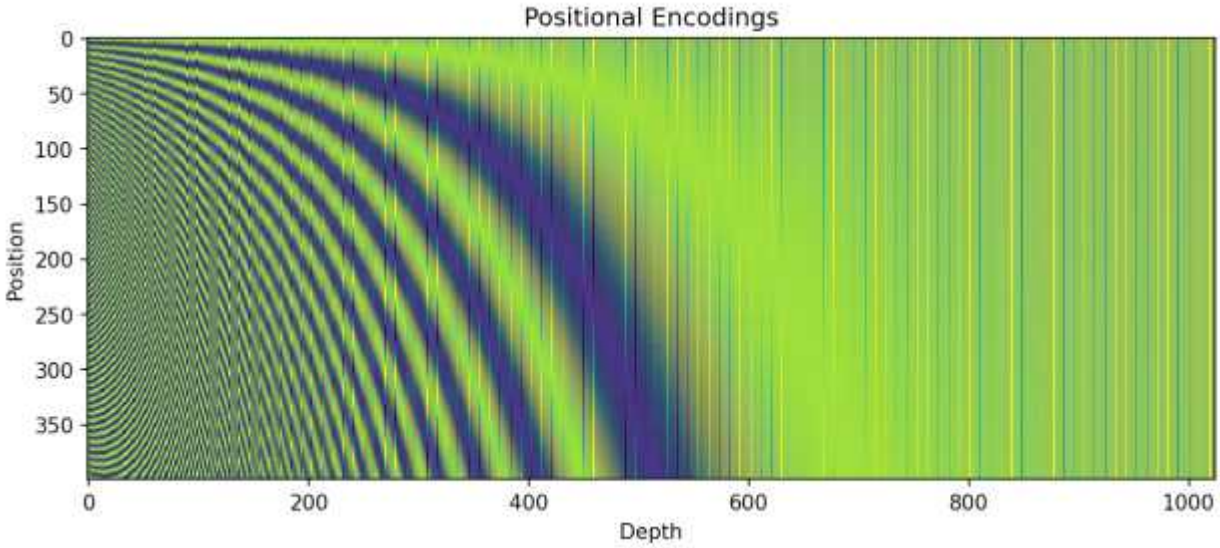
d_model=1024, sequence_length = 50




d_model=1024, sequence_length = 200




d_model=1024, sequence_length = 400



## 2.2 Source Code

(https://debuggercafe.com/text-generation-with-transformers/)

```
# Decoder Only Transformer
#       (https://debuggercafe.com/text-generation-with-transformers/)
# conda install matplotlib
# conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c nvidia
#       또는 conda install pytorch torchvision torchaudio cpuonly -c pytorch
# set  KMP_DUPLICATE_LIB_OK=TRUE

# python transformer_decoder.py train 2

importtorch
importtorch.nn as nn
importtorch.optim as optim
importtorch.nn.functional as F
fromtorch.utils.data importDataset,DataLoader
from collections import Counter
from tqdm import tqdm
import sys,math,warnings
import matplotlib.pyplot as plt
# Ignore Flash Attention compilation warnings
warnings.filterwarnings("ignore",message=".*not compiled with flash attention.*")
#=============================================================================
# [1/4] Data Preparation
#=============================================================================
SEQUENCE_LENGTH =16     # 문장내 최대 단어 갯수 크기를 16으로 설정
with open('book.txt','r')as file:
    text =file.read()
# Tokenize the text into words
words =text.split()
word_counts =Counter(words)
vocab =list(word_counts.keys())
vocab_size =len(vocab)
word_to_int ={word:i for i,word in enumerate(vocab)}
int_to_word ={i:word for word,i in word_to_int.items()}
samples =[words[i:i+SEQUENCE_LENGTH+1]for i in range(len(words)-SEQUENCE_LENGTH)]
print(f"\nvocab : 총 [{vocab_size}] 단어 중 앞 10개\n",vocab[:10])
print("\nword_to_int[0~10]")
for i,word in enumerate(word_to_int):
    print(f"[{word}],",end='\t')
    if i==9 :break
#=============================================================================
# My Dataset Class
class TextDataset(Dataset):
    def __init__(self,samples,word_to_int):
        self.samples =samples
        self.word_to_int =word_to_int
    def __len__(self):
        return len(self.samples)
    def __getitem__(self,idx):
        sample =self.samples[idx]
            # 첫문자에서 마지막 문자까지 input vector 만들기
        input_seq =torch.LongTensor([self.word_to_int[word]for word in sample[:-1]])
            # 둘째 문자부터 마지막 문자까지 target vector 만들기
        target_seq =torch.LongTensor([self.word_to_int[word]for word in sample[1:]])
        return input_seq,target_seq
#=============================================================================
#     Generate a square mask for the sequence. The masked positions are filled with
float('-inf').
#    Unmasked positions are filled with float(0.0).
def generate_square_subsequent_mask(sz,device):
         # sz크기의 정방형 upper triangular matrix 후 transpose (filled with 1 in the upper
triangle and 0 elsewhere)
```

```
    mask =(torch.triu(torch.ones(sz,sz,device=device))==1).transpose(0,1)
        # 0의 값 --> -inf 로,  1의 값 --> 0으로 바꿈
    mask =mask.float().masked_fill(mask ==0,float('-inf')).masked_fill(mask ==1,float(0.0))
    return mask
#=============================================================================
# Embedding 된 입력 벡터에 Positional Encoding 값을 더하여 반환
class PositionalEncoding(nn.Module):
    def __init__(self,max_len,d_model,dropout=0.1):
        """
:param max_len: Input length sequence.
:param d_model: Embedding dimension.
:param dropout: Dropout value (default=0.1)
        """
        super(PositionalEncoding,self).__init__()
        self.dropout =nn.Dropout(p=dropout)
        pe =torch.zeros(max_len,d_model)
        position =torch.arange(0,max_len,dtype=torch.float).unsqueeze(1)
        div_term =torch.exp(torch.arange(0,d_model,2).float()*(-math.log(10000.0)/d_model))
        pe[:,0::2]=torch.sin(position *div_term)
        pe[:,1::2]=torch.cos(position *div_term)
        pe =pe.unsqueeze(0)
        self.register_buffer('pe',pe)

    def forward(self,x):
        """
Inputs of forward function
:param x: the sequence fed to the positional encoder model (required).
Shape:
x: [sequence length, batch size, embed dim]
output: [sequence length, batch size, embed dim]
        """
        x =x +self.pe[:,:x.size(1)]
        return self.dropout(x)
#=============================================================================
class TextGenDecoder(nn.Module):
    def __init__(self,vocab_size,embed_dim,num_layers,num_heads):
        super(TextGenDecoder,self).__init__()
        self.pos_encoder =PositionalEncoding(max_len=SEQUENCE_LENGTH,d_model=embed_dim)
        self.emb =nn.Embedding(vocab_size,embed_dim)
        self.decoder_layer =nn.TransformerDecoderLayer(
            d_model=embed_dim,
            nhead=num_heads,
            batch_first=True
        )
        self.decoder =nn.TransformerDecoder(
            decoder_layer=self.decoder_layer,
            num_layers=num_layers,
        )
        self.linear =nn.Linear(embed_dim,vocab_size)
        self.dropout =nn.Dropout(0.2)

    # Positional encoding is required. Else the model does not learn.
    def forward(self,x):
        emb =self.emb(x)

        # Generate input sequence mask with shape (SEQUENCE_LENGTH, SEQUENCE_LENGTH)
        input_mask =generate_square_subsequent_mask(x.size(1),x.device)

        x =self.pos_encoder(emb)
        x =self.decoder(x,memory=x,tgt_mask=input_mask,memory_mask=input_mask)
        x =self.dropout(x)
        out =self.linear(x)
        return out

#=============================================================================
# draw loss changes
```

```
def draw_loss_changes(allEpochLosses):
    plt.plot(allEpochLosses,marker='o',linestyle='-',color='b')
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid(True)
    plt.show()

#================================================================================
# Training
def trainDecoder(model,totalEpochs,dataloader,criterion,modelPath,printInterval=10):
    device =torch.device("cuda"if torch.cuda.is_available()else "cpu")
    debugFlag =False
    if debugFlag :  # 각 파라미터가 gpu/cpu 중 뭘 사용하는지 확인
        for name,param in model.named_parameters():
            print(f"{name}is on {param.device}")
    model.train()   # 학습 모드로 설정

    print('\n','='*80,'\n*** Training Starting......')
    allEpochLosses =[]
    totalDataSize =len(dataloader)
    for epochNo in tqdm(range(totalEpochs)):
        running_loss =0
        for dataNo,(input_seq,target_seq)in enumerate(dataloader):
            input_seq,target_seq =input_seq.to(device),target_seq.to(device)

            outputs =model(input_seq)
            if epochNo ==0 and dataNo ==0  :
                print('-'*80)
                print(f'\nEpoch #[{epochNo}]')
                print('input 크기',input_seq.shape)
                print('input[:3] 값:',input_seq[:3])
                print('\ntarget 크기',target_seq.shape)
                print('target[:3] 값: ',target_seq[:3])
                print('\noutputs 크기: ',outputs.shape,'\n')
            target_seq =target_seq.contiguous().view(-1)
            outputs =outputs.view(-1,vocab_size)
            if dataNo %printInterval ==0  :
                 print(f'\n---------- epochNo: [{epochNo}], dataNo [{dataNo}of {totalDataSize}]
----------')
                print('***outputs[0]:',outputs[0].detach().cpu().numpy())

            loss =criterion(outputs,target_seq.view(-1))
            if dataNo %printInterval ==0  :
                print(f'\t*** [{dataNo}]번째 data에 대한 loss:',loss.detach().cpu().numpy())


            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            running_loss +=loss.detach().cpu().numpy()

        epoch_loss =running_loss /len(dataloader)
        allEpochLosses.append(epoch_loss)
        print('\n','='*80)
        print(f"======== Epoch {epochNo}loss: {epoch_loss:.3f}========")
        print('='*80,'\n')

    draw_loss_changes(allEpochLosses)
    torch.save(model.state_dict(),modelPath)

    return allEpochLosses
#================================================================================
# 입력 문장을 정수 벡터로 변환
def get_integer_vector(text):
    words =text.split()
```

```
                input_seq       =torch.LongTensor([word_to_int[word]for       word       in
words[-SEQUENCE_LENGTH:]]).unsqueeze(0)
    return input_seq
#================================================================================
# 예측 문장 생성
def sample_next(predictions):
    """
Greedy sampling.
    """
    # Greedy approach.
    probabilities =F.softmax(predictions[:,-1,:],dim=-1).cpu()
    next_token =torch.argmax(probabilities)
    return int(next_token.cpu())
#================================================================================
# Prediction
def predict(model,sentence,totalPreditedWordNo):
    model.eval()
    sample =sentence
    for i in range(totalPreditedWordNo):
        int_vector =get_integer_vector(sample)
        if len(int_vector)>=SEQUENCE_LENGTH -1:
            break
        input_tensor =int_vector.to(device)

        with torch.no_grad():
            predictions =model(input_tensor)

        next_token_ndx =sample_next(predictions)
        sample +=''+int_to_word[next_token_ndx]
    return(sample)
#================================================================================
# Model feature printing
def displayModelFeature(model):
    if showModelFlag :
        print('-'*80)
        print('Decoder Model 구조')
        print(model)
        print('-'*80)
        # Total parameters and trainable parameters.
        total_params =sum(p.numel()for p in model.parameters())
        print(f"\n{total_params:,}total parameters.")
        total_trainable_params =sum(
            p.numel()for p in model.parameters()if p.requires_grad)
        print(f"{total_trainable_params:,}training parameters.\n")

#================================================================================
# Gpu status printing
def displayGpuStatus(showFlag):
    totalGpuNum =torch.cuda.device_count()
    if showFlag :
        print("="*80)
        if totalGpuNum  >0 :
            if torch.cuda.is_available():
                torch.cuda.empty_cache()
            print("Available GPU Devices: ")
            for i in range(totalGpuNum):
                print("[[[ Device [%d]: %s]]]"%(i,torch.cuda.get_device_properties(i)))
            print("-"*50)
            print("현재 사용중인 GPU #: ",torch.cuda.current_device())
            print('Allocated:',round(torch.cuda.memory_allocated(0)/1024**3,1),'GB')
            print('Cached:   ',round(torch.cuda.memory_reserved(0)/1024**3,1),'GB')
        else :
            print("NO Gpu exists..... Will use the CPU")
        print("="*80)
    return totalGpuNum
```

```
#================================================================================
# [2/4] dataset 준비
#================================================================================
BATCH_SIZE =32
dataset =TextDataset(samples,word_to_int)
dataloader =DataLoader(
    dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
)
print('\n*** input, target dataset[0]:\n',dataset[0])
print('\n*** input, target dataset[1]:\n',dataset[1])
modelPath ='./decoder.pth'
#================================================================================
# [3/4] Decoder 모델 준비
#================================================================================
totalEpochs =100
learning_rate =0.001
device =torch.device('cuda'if torch.cuda.is_available()else 'cpu')
print(f'\n*** device = {device}\n')
printInterval =10 if device =='cpu'else 100

model =TextGenDecoder(
    vocab_size=vocab_size,
    embed_dim=512,
    num_layers=6,
    num_heads=4,
).to(device)
criterion =nn.CrossEntropyLoss()
optimizer =optim.Adam(model.parameters(),lr=learning_rate)
showModelFlag =True
if showModelFlag ==True:
    displayModelFeature(model)

#================================================================================
# [4/4] main, train or predict
#================================================================================
if __name__=="__main__":
    print('\n','='*80)
    print('Usage [Training]: python decoder.py   train   totalEpochNo[예:10]')
    print('Usage [Prediction]: python decoder.py   predict   totalPreditedWordNo[예:100]')
    print('*** ',displayGpuStatus(True),'Gpu(s) will be used.\n')

    if len(sys.argv)>=3 :
            #-------------------------------- 학습
        if 'train'in sys.argv[1]and int(sys.argv[2])>0:
            totalEpochs =int(sys.argv[2])
                                                                                allEpochLosses
=trainDecoder(model,totalEpochs,dataloader,criterion,modelPath,printInterval)
            # Test prediction using a sentence
        elif 'predict'in sys.argv[1]and int(sys.argv[2])>0:
            totalPreditedWordNo =int(sys.argv[2])
            state_dict =torch.load(modelPath)
            # Update the current model state with the loaded state dictionary
            model.load_state_dict(state_dict)
            model =model.to(device)
            print("*** Model loaded successfully from ",modelPath,'\n')

            if showModelFlag :
                displayModelFeature(model)

            while True:
                query_sentence =input("\n*** Type a sentence or 'quit' to stop: ")
                if 'quit'in query_sentence :
                    break
                #print("Predicting for sentence:", test_sentence)
```

```
            print("===> query sentence:",query_sentence)
            predicted_output =predict(model,query_sentence,totalPreditedWordNo)
            print("===> Predicted output:",predicted_output,'\n')
```

## 2.3 실행결과 (Train, Prediction)

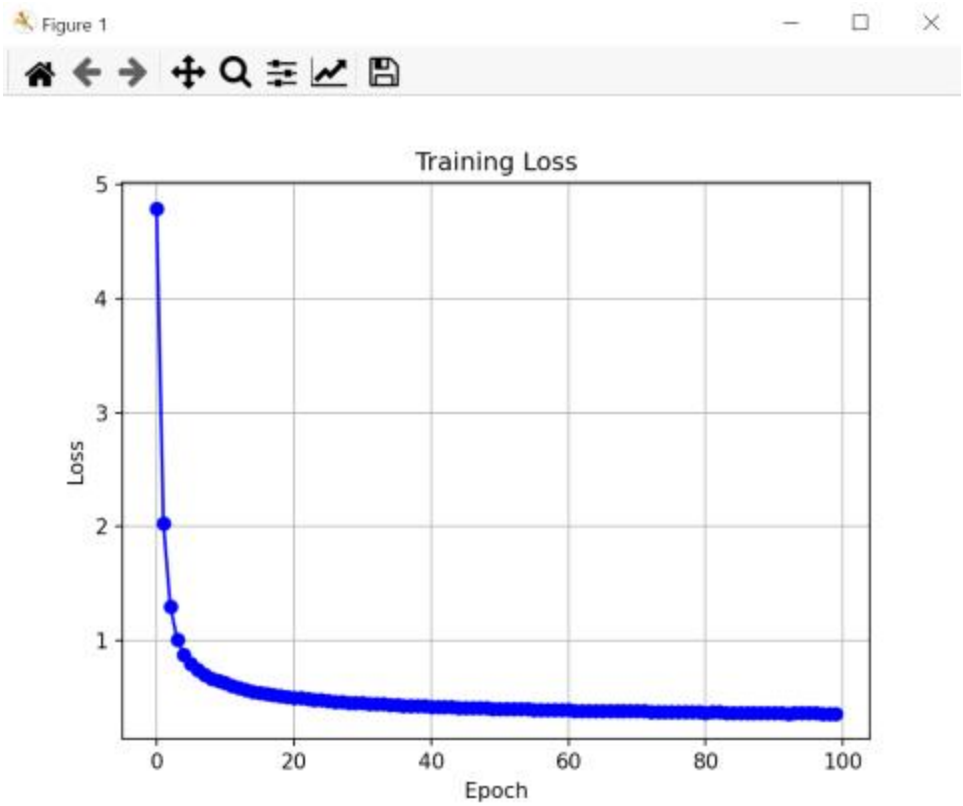
```
(word2vec) D:\myprojects\nlp\transformer>python transformer_decoder.py train 100

vocab : 총 [6790] 단어 중 앞 10개
 ['You', 'will', 'rejoice', 'to', 'hear', 'that', 'no', 'disaster', 'has', 'accompanied']

word_to_int[0~10]
[You], [will], [rejoice], [to], [hear], [that], [no], [disaster], [has], [accompanied],
*** input, target dataset[0]:
 (tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]), tensor([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10, 11, 12, 13, 14, 15, 16]))

*** input, target dataset[1]:
 (tensor([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]), tensor([ 2,  3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14, 15, 16, 17]))

*** device = cuda

--------------------------------------------------------------------------------
Decoder Model 구조
TextGenDecoder(
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (emb): Embedding(6790, 512)
  (decoder_layer): TransformerDecoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
    )
    (multihead_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.1, inplace=False)
    (dropout2): Dropout(p=0.1, inplace=False)
    (dropout3): Dropout(p=0.1, inplace=False)
  )
  (decoder): TransformerDecoder(
    (layers): ModuleList(
      (0-5): 6 x TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (linear): Linear(in_features=512, out_features=6790, bias=True)
```

```
  (dropout): Dropout(p=0.2, inplace=False)
)
--------------------------------------------------------------------------------

36,387,974 total parameters.
36,387,974 training parameters.


 ================================================================================
Usage [Training]: python decoder.py    train    totalEpochNo[예:10]
Usage [Prediction]: python decoder.py    predict    totalPreditedWordNo[예:100]
================================================================================
Available GPU Devices:
[[[ Device [0]: _CudaDeviceProperties(name='NVIDIA GeForce GTX 1080 Ti', major=6, minor=1,
total_memory=11263MB, multi_processor_count=28) ]]]
------------------------------------------------
현재 사용중인 GPU #:  0
Allocated: 0.1 GB
Cached:    0.1 GB
================================================================================
***  1  Gpu(s) will be used.


 ================================================================================
*** Training Starting.....
  0%|                                                                                                  | 0/100
[00:00<?, ?it/s]--------------------------------------------------------------------------------

Epoch #[0]
input 크기] torch.Size([32, 16])
input[:3] 값: tensor([[ 260, 2134,  531,   28,  354, 2135,   23, 1316, 1337, 2136, 2137,   28,
         1099,  412, 1292,   27],
        [4946, 4947,   80,  240, 4948, 2890,   31, 4949,   23,  490,  147,  211,
         4950, 4951,   27,   23],
        [ 571, 2283, 3845, 6536,  152,  493,  404, 5077, 6537,   23,  405,    3,
          232,  263,  137, 6538]], device='cuda:0')

target 크기] torch.Size([32, 16])
target[:3] 값:  tensor([[2134,  531,   28,  354, 2135,   23, 1316, 1337, 2136, 2137,   28, 1099,
         412, 1292,   27, 2138],
        [4947,   80,  240, 4948, 2890,   31, 4949,   23,  490,  147,  211, 4950,
         4951,   27,   23, 4857],
        [2283, 3845, 6536,  152,  493,  404, 5077, 6537,   23,  405,    3,  232,
          263,  137, 6538,   27]], device='cuda:0')

outputs 크기:  torch.Size([32, 16, 6790])


---------- epochNo: [0], dataNo [0 of 969] ----------
***outputs[0]: [-1.0861089  -0.34837022  0.53277653 ... -0.9499545  -0.12830055
  0.90849656]
        *** [0]번째 data에 대한 loss: 9.077804

---------- epochNo: [0], dataNo [100 of 969] ----------
***outputs[0]: [ 1.3261777  2.0043547 -7.0943227 ... -4.59713    -4.1754107 -6.3519225]
        *** [100]번째 data에 대한 loss: 6.8431497

---------- epochNo: [0], dataNo [200 of 969] ----------
***outputs[0]: [ 0.6198843  3.3357806 -3.3884528 ... -4.358764   -4.319337   -5.6916246]
        *** [200]번째 data에 대한 loss: 6.410752

---------- epochNo: [0], dataNo [300 of 969] ----------
***outputs[0]: [-1.1420043  1.298211  -4.8781104 ... -4.5287395 -2.5426219 -5.237867 ]
        *** [300]번째 data에 대한 loss: 5.574061

---------- epochNo: [0], dataNo [400 of 969] ----------
```

```
***outputs[0]: [-1.090288   -1.219418   -2.7754295  ... -2.490411    0.50771165
 -3.98669   ]
        *** [400]번째 data에 대한 loss: 5.232719

---------- epochNo: [0], dataNo [500 of 969] ----------
***outputs[0]: [ 0.58102435  0.64548516 -3.6217003  ... -3.7999682  -3.5443857
 -5.1272492 ]
        *** [500]번째 data에 대한 loss: 4.498461

---------- epochNo: [0], dataNo [600 of 969] ----------
***outputs[0]: [-2.5544684  2.3502975 -5.932676  ... -1.8427486 -2.05928   -6.8918233]
        *** [600]번째 data에 대한 loss: 3.734997

---------- epochNo: [0], dataNo [700 of 969] ----------
***outputs[0]: [-2.0071793  3.0888677 -5.68414   ... -3.7020614 -5.2527966 -8.305013 ]
        *** [700]번째 data에 대한 loss: 3.5377414

---------- epochNo: [0], dataNo [800 of 969] ----------
***outputs[0]: [ 0.5470243  1.867981  -4.6772013 ... -2.9333136 -4.428924  -6.128312 ]
        *** [800]번째 data에 대한 loss: 3.2594347

---------- epochNo: [0], dataNo [900 of 969] ----------
***outputs[0]: [-2.551892  -4.3637977 -5.281256  ... -2.2248583 -5.5458674 -5.2576227]
        *** [900]번째 data에 대한 loss: 2.7963686

 ================================================================================
======== Epoch 0 loss: 4.789 ========

================================================================================

  1%|▉                                                                                | 1/100
[00:33<54:47, 33.21s/it]
---------- epochNo: [1], dataNo [0 of 969] ----------
***outputs[0]: [-2.186807   2.4395247 -2.9326239 ... -2.8534584  1.3486004 -6.1811476]
        *** [0]번째 data에 대한 loss: 2.67188

---------- epochNo: [1], dataNo [100 of 969] ----------
***outputs[0]: [-2.2192128  0.8328568 -8.526669  ... -8.606551  -5.3712287 -8.034913 ]
        *** [100]번째 data에 대한 loss: 2.321597

---------- epochNo: [1], dataNo [200 of 969] ----------
***outputs[0]: [ 0.02637544 -0.8418565  -4.8529544  ... -6.1378384  -3.9743657
 -7.0091314 ]
        *** [200]번째 data에 대한 loss: 2.3624792

---------- epochNo: [1], dataNo [300 of 969] ----------
***outputs[0]: [-0.38317975  4.645627   -6.849756   ... -8.546732   -6.006018
 -7.93426   ]
        *** [300]번째 data에 대한 loss: 2.4673202

---------- epochNo: [1], dataNo [400 of 969] ----------
***outputs[0]: [ 0.23307392 -4.0915422  -2.9277515  ...  4.3009286  -1.2285931
 -5.213801  ]
        *** [400]번째 data에 대한 loss: 2.2409756

---------- epochNo: [1], dataNo [500 of 969] ----------
***outputs[0]: [-1.1112174 -1.2388387 -2.9004495 ...  3.5497732  1.0115964 -3.6816416]
        *** [500]번째 data에 대한 loss: 2.1637707

---------- epochNo: [1], dataNo [600 of 969] ----------
***outputs[0]: [-3.1266725 -2.2506049 -8.77348   ... -5.7513137 -6.4414372 -6.0197024]
        *** [600]번째 data에 대한 loss: 1.9001055

---------- epochNo: [1], dataNo [700 of 969] ----------
***outputs[0]: [-2.161523   -0.24332418 -7.1364827  ... -3.512435   -0.5284645
 -3.9911094 ]
```

```
        *** [700]번째 data에 대한 loss: 2.0788023

---------- epochNo: [1], dataNo [800 of 969] ----------
***outputs[0]: [ 2.7411945  2.741202  -1.9532055 ... -2.9264987 -8.788032  -1.2980127]
        *** [800]번째 data에 대한 loss: 1.6578261

---------- epochNo: [1], dataNo [900 of 969] ----------
***outputs[0]: [-0.910451  -2.2996292 -6.220722  ... -6.0931535 -1.4642068 -4.248584 ]
        *** [900]번째 data에 대한 loss: 1.6988271

 ================================================================================
======== Epoch 1 loss: 2.035 ========

================================================================================

  2%|██                                                                  | 2/100
[01:06<54:42, 33.50s/it]
99%|████████████████████████████████████████████████████████████████████
████████████████████████████████████████████▌| 99/100 [53:15<00:31, 31.99s/it]
---------- epochNo: [99], dataNo [0 of 969] ----------
***outputs[0]: [-0.31403175 -0.08858681 -2.499173   ...  0.18549322 -1.2927423
 -3.600165  ]
        *** [0]번째 data에 대한 loss: 0.3121465

---------- epochNo: [99], dataNo [100 of 969] ----------
***outputs[0]: [-4.4494567  2.1241283 -2.2779741 ... -2.472934  -1.1369834 -2.4487207]
        *** [100]번째 data에 대한 loss: 0.34908825

---------- epochNo: [99], dataNo [200 of 969] ----------
***outputs[0]: [ 1.3604805 -1.0227257 -3.8957782 ... -0.7976928 -1.6082261 -6.315931 ]
        *** [200]번째 data에 대한 loss: 0.31011933

---------- epochNo: [99], dataNo [300 of 969] ----------
***outputs[0]: [-0.8759607  -0.79515725 -2.7890873  ... -2.4043047  -5.5015416
 -3.3856    ]
        *** [300]번째 data에 대한 loss: 0.38624457

---------- epochNo: [99], dataNo [400 of 969] ----------
***outputs[0]: [-0.61345464 -4.126785   -4.4495864  ... -9.494722   -5.758465
 -6.153773  ]
        *** [400]번째 data에 대한 loss: 0.2446664

---------- epochNo: [99], dataNo [500 of 969] ----------
***outputs[0]: [-1.5889344  5.3392143 -4.035906  ... -6.719337   0.8126576 -4.356104 ]
        *** [500]번째 data에 대한 loss: 0.35023996

---------- epochNo: [99], dataNo [600 of 969] ----------
***outputs[0]: [ 2.009325  -3.6023393 -0.9653509 ... -1.0695437 -1.2056429 -2.645245 ]
        *** [600]번째 data에 대한 loss: 0.39789605

---------- epochNo: [99], dataNo [700 of 969] ----------
***outputs[0]: [-3.1229038  0.526892  -4.545939  ... -4.0135374 -4.5330973 -4.5231695]
        *** [700]번째 data에 대한 loss: 0.34770235

---------- epochNo: [99], dataNo [800 of 969] ----------
***outputs[0]: [-1.3588071  -0.86897856 -1.303202   ... -1.2703942  -1.5510566
 -2.6297438 ]
        *** [800]번째 data에 대한 loss: 0.35365683

---------- epochNo: [99], dataNo [900 of 969] ----------
***outputs[0]: [-11.276419    4.523317   -0.6609094 ...  -6.364531   -0.6709239
  -4.647649 ]
        *** [900]번째 data에 대한 loss: 0.3970822

 ================================================================================
======== Epoch 99 loss: 0.367 ========
```

```
================================================================================
100%|████████████████████████████████████████████████████████████████████████████
████████████████████████████████████| 100/100 [53:47<00:00, 32.27s/it]
```

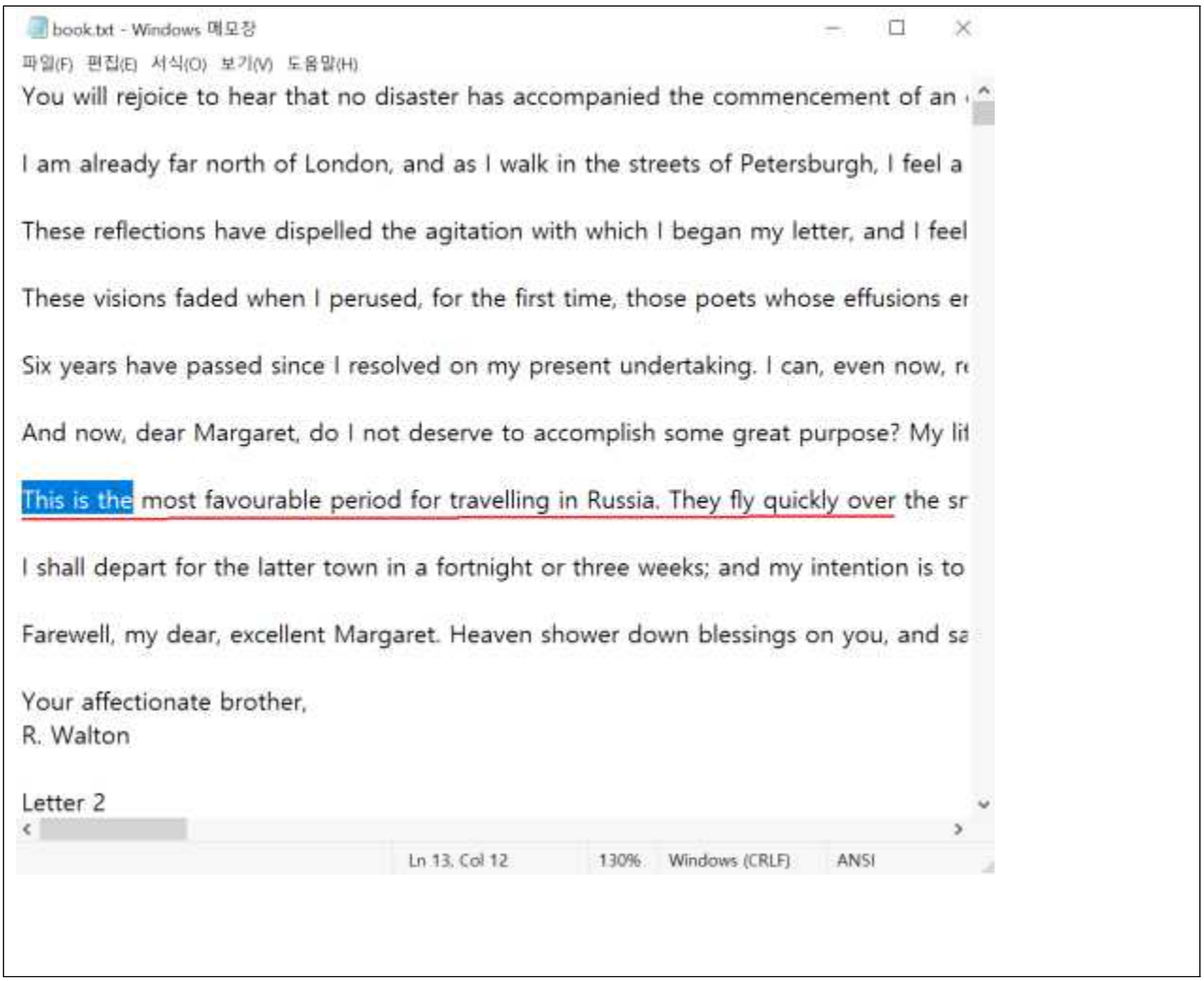



```
(word2vec) D:\myprojects\nlp\transformer>dir
 D 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 147D-5B3A

 D:\myprojects\nlp\transformer 디렉터리

2024-04-07  오후 09:23    <DIR>          .
2024-04-07  오후 09:23    <DIR>          ..
2024-03-02  오후 05:27           173,527 book.txt
2024-04-08  오후 12:57       145,631,406 decoder.pth
2024-04-07  오후 02:47       147,209,868 decoder_only.pth
2024-04-07  오후 04:42            16,644 decoder_only.py
2024-04-03  오후 02:55               557 fastapi_test.py
2024-03-28  오후 09:56               710 positional_encoding.py
2024-04-01  오후 08:01             6,900 self_attention.py
2024-04-06  오후 06:57       206,081,402 trained.pth
2024-03-28  오후 09:42             9,385 transformer.py
2024-04-07  오후 09:46            14,262 transformer_decoder.py
2024-04-07  오후 06:38             5,304 transformer2.py
2024-04-03  오후 02:55    <DIR>          __pycache__
              11개 파일         499,149,965 바이트
               3개 디렉터리  262,432,382,976 바이트 남음

(word2vec) D:\myprojects\nlp\transformer>

********************************************************************************
                        PREDICTION TEST (다음 10개의 단어)
********************************************************************************
```

```
(word2vec) D:\myprojects\nlp\transformer>python transformer_decoder.py predict 10

vocab : 총 [6790] 단어 중 앞 10개
 ['You', 'will', 'rejoice', 'to', 'hear', 'that', 'no', 'disaster', 'has', 'accompanied']

word_to_int[0~10]
[You], [will], [rejoice], [to], [hear], [that], [no], [disaster], [has], [accompanied],
*** input, target dataset[0]:
 (tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]), tensor([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10, 11, 12, 13, 14, 15, 16]))

*** input, target dataset[1]:
 (tensor([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]), tensor([ 2,  3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14, 15, 16, 17]))

*** device = cuda

--------------------------------------------------------------------------------
Decoder Model 구조
TextGenDecoder(
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (emb): Embedding(6790, 512)
  (decoder_layer): TransformerDecoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
    )
    (multihead_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.1, inplace=False)
    (dropout2): Dropout(p=0.1, inplace=False)
    (dropout3): Dropout(p=0.1, inplace=False)
  )
  (decoder): TransformerDecoder(
    (layers): ModuleList(
      (0-5): 6 x TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (linear): Linear(in_features=512, out_features=6790, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

```
--------------------------------------------------------------------------------

36,387,974 total parameters.
36,387,974 training parameters.


 ================================================================================
Usage [Training]: python decoder.py    train    totalEpochNo[예:10]
Usage [Prediction]: python decoder.py    predict    totalPreditedWordNo[예:100]
================================================================================
Available GPU Devices:
[[[ Device [0]: _CudaDeviceProperties(name='NVIDIA GeForce GTX 1080 Ti', major=6, minor=1,
total_memory=11263MB, multi_processor_count=28) ]]]
------------------------------------------------
현재 사용중인 GPU #:  0
Allocated: 0.1 GB
Cached:    0.1 GB
================================================================================
***  1  Gpu(s) will be used.

*** Model loaded successfully from  ./decoder.pth

--------------------------------------------------------------------------------
Decoder Model 구조
TextGenDecoder(
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (emb): Embedding(6790, 512)
  (decoder_layer): TransformerDecoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
    )
    (multihead_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.1, inplace=False)
    (dropout2): Dropout(p=0.1, inplace=False)
    (dropout3): Dropout(p=0.1, inplace=False)
  )
  (decoder): TransformerDecoder(
    (layers): ModuleList(
      (0-5): 6 x TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512, out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
    )
```

```
  )
  (linear): Linear(in_features=512, out_features=6790, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
----------------------------------------------------------------------------------

36,387,974 total parameters.
36,387,974 training parameters.


*** Type a sentence or 'quit' to stop: this is the
===> query sentence: this is the
===> Predicted output: this is the scene of my labours. It was a place fitted for


*** Type a sentence or 'quit' to stop: this is the most
===> query sentence: this is the most
===> Predicted output: this is the most favourable period for travelling in Russia. They fly quickly
over


*** Type a sentence or 'quit' to stop: this is
===> query sentence: this is
===> Predicted output: this is your present temper, my friend, you will perhaps be glad


*** Type a sentence or 'quit' to stop: i am already
===> query sentence: i am already
===> Predicted output: i am already commenced, and I expected the letter daily which was to


*** Type a sentence or 'quit' to stop:
```


book.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
You will rejoice to hear that no disaster has accompanied the commencement of an
I am already far north of London, and as I walk in the streets of Petersburgh, I feel a
These reflections have dispelled the agitation with which I began my letter, and I feel
These visions faded when I perused, for the first time, those poets whose effusions er
Six years have passed since I resolved on my present undertaking. I can, even now, re
And now, dear Margaret, do I not deserve to accomplish some great purpose? My lif
This is the most favourable period for travelling in Russia. They fly quickly over the sr
I shall depart for the latter town in a fortnight or three weeks; and my intention is to
Farewell, my dear, excellent Margaret. Heaven shower down blessings on you, and sa
Your affectionate brother,
R. Walton
Letter 2
Ln 13, Col 12
130%
Windows (CRLF)
ANSI

## ※ [Questions]

# 1. 입력 시퀀스 길이를 조정하려면 어떻게 해야 하는가?

# 2. target vector 시퀀스 길이가 현재는 입력과 같은데, target sequence 길이를 1(다음단어 1개)로 변경하려면 어떻게 해야 하는가?

## ※ [실습 문제]

# 3. 단어 사전에 등록되지 않은 단어를 query_sentence에 사용하면 어떻게 되는가? 미등록 단어(token)가 입력된 경우 <unk> 특수기호를 추가하도록 소스를 수정 해 보시오.

```
*** Type a sentence or 'quit' to stop: qqq rrr
===> query sentence: qqq rrr
Traceback (most recent call last):
  File "D:\myprojects\nlp\transformer\transformer1.py", line 345, in <module>
    predicted_output = predict(model, query_sentence, totalPreditedWordNo)
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "D:\myprojects\nlp\transformer\transformer1.py", line 221, in predict
    int_vector = get_integer_vector(sample)
                 ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "D:\myprojects\nlp\transformer\transformer1.py", line 201, in get_integer_vector
    input_seq = torch.LongTensor([word_to_int[word] for word in
words[-SEQUENCE_LENGTH:]]).unsqueeze(0)
                                 ~~~~~~~~~~~^^^^^^
KeyError: 'qqq'
```

# 4. 학습시 사용된 입력 벡터 시퀀스의 길이가 한개의 값으로 정해져 있는데, 이는 주어진 학습문장에 대해, 지정된 길이보다 짧은 시퀀스에 대한 학습기회를 놓치는 낭비적인 면이 있다. 주어진 입력 문장에 대해, 그보다 짧은 다수의 가변길이 데이터에 대해서도 학습하게 하려면, 학습 배치당 최대길이 입력시퀀스를 매 bqtch 데이터로부터 정하고, 그 길이값 보다 짧은 입력 문장의 경우 남는 자리의 벡터값을 <pad> 값으로 채워 넣어서 학습하면 된다. 이 때, padded된 시퀀스에 대한 loss 계산도 그에 맞추어야 한다.