llm---bert

warning) 오류들이 있을 수 있음

encoder로 동의어관계 파악/추출, decoder는 생성형이 아니므로 필수적이지 않음

만드는중

1) 데이터셋 : hugging face의 ("glue", "mrpc") , sentence_1 , sentence_2 사이의 동의어 관계 분석을 위한 구조를 가지고 있음

['sentence1', 'sentence2', 'label', 'idx']

2) 데이터시각화

i) label distribution : Not equivalent(=0)이 equivalent(=1)에 비해 대략 절반 정도다.-> calss mismatch이므로 bce_loss에서 equivalent label의 경우, pos_weight을 0.5로 조정해 loss 수식에서의 영향력을 절반 정도로 줄인다

ii) sentence length distribution : 가지각색이다. zero padding은 64로 맞추면 되지만, sentence length의 frequency가 동일하지 않고 대략 sentence length 20~25를 중심으로 몰려있다. 어떤 데이터는 주변부 등 자잘한 의미를 더 많이 담고 있을 가능성이 높다.

3) 토큰화. pretrained bert model로 tokenizer를 생성해 문장을 토큰화한다.

4) hyperparameter에서,
embedding_dim = 64(=sentence_length) * 8(=num_heads)이다.

5) pre_process

```
class Pre_process(nn.Module):
    def __init__(self ,num_heads , embedding_dim , vocab_size = 40000 , max_length = 64, dropout=0.1):
        super().__init__()
        self.embed_size = embedding_dim
        self.max_length = max_length
        self.word_embedding = nn.Embedding(vocab_size, embedding_dim)
        self.position_embedding = nn.Parameter(torch.randn(1 , max_length , embedding_dim))
        self.dropout = nn.Dropout(dropout)
    def run(self , input , token_type_ids , attention_mask):
        batch_num , seq_length = input.shape
        z = self.word_embedding(input) # z.shape : batch_num , sequence_length , embedding_dim
        positional_embed = self.position_embedding.expand(batch_num , self.max_length , self.embed_size)
        # 단어 임베딩 + position + 문장 소속(binary)
        out = z + positional_embed + token_type_ids.unsqueeze(2).expand(batch_num,self.max_length,self.embed_size) + attention_mask.unsqueeze(2).expand(batch_num,self.max_length,self.embed_size)
        out = self.dropout(out)
        return out # out.shape : batch_num , sequence_length , embed_size
```

단어 임베딩 텐서와 positional embedding 텐서, 문장 소속이 어디인지 구분하는 텐서, zero padding에 대해 attention을 하지 않기 위해 masking처리를 해주는 텐서 총 4가지에 input을 더한다.

i) positional embedding의 경우, cosine등을 쓰기도 하는데 이 프로젝트에서는 학습 가능한 변수로 학습.
ii) 추가 성능향상을 위해 시도해볼 수 있는 방법 중 하나가 embedding하기 전에 필요없는 문법에 필요한 조사 등을 제거하는 작업이다.

6) Attention

```
class MultiHeadAttention(nn.Module):
  def __init__(self, num_heads , embedding_dim , input_size = 16):
    super().__init__()
    self.embedding_dim = embedding_dim
    self.num_heads = num_heads
    self.head_dim = embedding_dim // num_heads
    #self.input_size = input_size

    self.q = nn.Linear(embedding_dim , embedding_dim)
    self.k = nn.Linear(embedding_dim , embedding_dim)
    self.v = nn.Linear(embedding_dim , embedding_dim)

    self.fc = nn.Linear(embedding_dim , embedding_dim)

  def go(self , x):

    batch_size = x.shape[0]

    q = self.q(x) # batch_num , sequence_length , embed_size : 16 , 16 , 128
    k = self.k(x)
    v = self.v(x)

    q = q.view(batch_size , -1 , self.num_heads , self.head_dim).transpose(1,2) # batch , num_head, 16 , 16*8//8 = 16
    k = k.view(batch_size , -1 , self.num_heads , self.head_dim).transpose(1,2)
    v = v.view(batch_size , -1 , self.num_heads , self.head_dim).transpose(1,2)

    # Scaled Dot-Product Attention
    attention_score = (q @ k.transpose(3,2)) / (self.head_dim ** 0.5)
    attention_score = torch.softmax(attention_score, dim=-1)
    attention = torch.matmul(attention_score , v)
    attention = attention.transpose(1,2).contiguous().view(batch_size , -1 , self.embedding_dim) # batch , 8 , sequence_length , 16*8//8 = 16
    attention = attention.reshape(x.shape[0] , x.shape[1] , x.shape[2])
    output = self.fc(attention)

    return output # batch , sequence_length , 16*8
```

i) batch_num , num_heads를 기준으로 병렬적으로 (sequence_length,sequence_length)에 대해 attention을 구한다.

7) Encoder block

preprocess -> attention -> encoder_layer를 거치면 된다.

encoder layer는 bert의 경우, layer_norm -> linear -> relu ->...등을 거친다.

8) 결과

(check point , scheduling) 없음 , 모델훈련에는 train_dataset만 사용

epoch 99 , train_loss : 0.4397… ,
model.eval()로 test_dataset 결과:
i) accuracy : 0.6649…
ii)f1 : 0.7987…

class 불균형으로 인해 accuracy와 f1 차이가 큰걸로 보인다.
train_loss는 bce_loss인데 꽤 크다.

loss에 pos_weight를 추가한것만으로도 train_loss가 0.59 부근에서 0.4397…로 크게 줄었다.

9) 추가 실험해볼만한 요소들

i) sequence length의 차이로 인한 데이터마다의 담고있는 정보의 양적 차이
ii) hyperparameter 조정
iii) 모델 크기 조정
iv) 데이터 증강/다른 데이터 추가
v) pretrained model과 함께 사용