Chapter 14

트랜스포머 모델

학습 목표

- **트랜스포머 개념 및 동작 원리 이해**: 트랜스포머 아키텍처의 기본 개념과 작동 방식을 명확히 이해하고 특히, 주목(attention) 메커니즘과 인코더-디코더 구조의 역할을 설명할 수 있다.
- **트랜스포머 구현 능력**: 트랜스포머 모델을 실제로 구현하고, 다양한 프레임워크(예: TensorFlow, PyTorch)를 활용하여 모델을 훈련시키고 평가할 수 있다.
- **자연어 처리(NLP) 분야에서의 응용 사례 분석**: 트랜스포머 모델이 자연어 처리에서 어떻게 활용되는지 구체적인 사례를 통해 설명할 수 있으며, 예를 들어, 기계 번역, 텍스트 요약, 감정 분석 등 다양한 응용 분야를 제시할 수 있다.

14.1 트랜스포머란 무엇인가요?

- 트랜스포머는 자연어 처리(NLP) 분야에서 혁신을 일으킨 모델입니다.
- 2017년 구글에서 발표한 논문 "Attention is All You Need"에서 처음 소개되었으며, 이후 다양한 응용 분야에서 뛰어난 성능을 보여주고 있습니다.
- 트랜스포머는 주로 번역, 텍스트 생성, 감정 분석 등에서 사용됩니다.

주요 특징

- **자기 주의 메커니즘(Self-Attention):** 입력 시퀀스의 각 단어가 다른 단어들과 얼마나 관련이 있는지를 학습합니다.
- **병렬 처리:** RNN과 달리 입력 데이터를 동시에 처리할 수 있어 학습 속도가 빠릅니다.
- **스케일러빌리티:** 대규모 데이터와 모델에 잘 맞습니다.

트랜스포머의 구성 요소

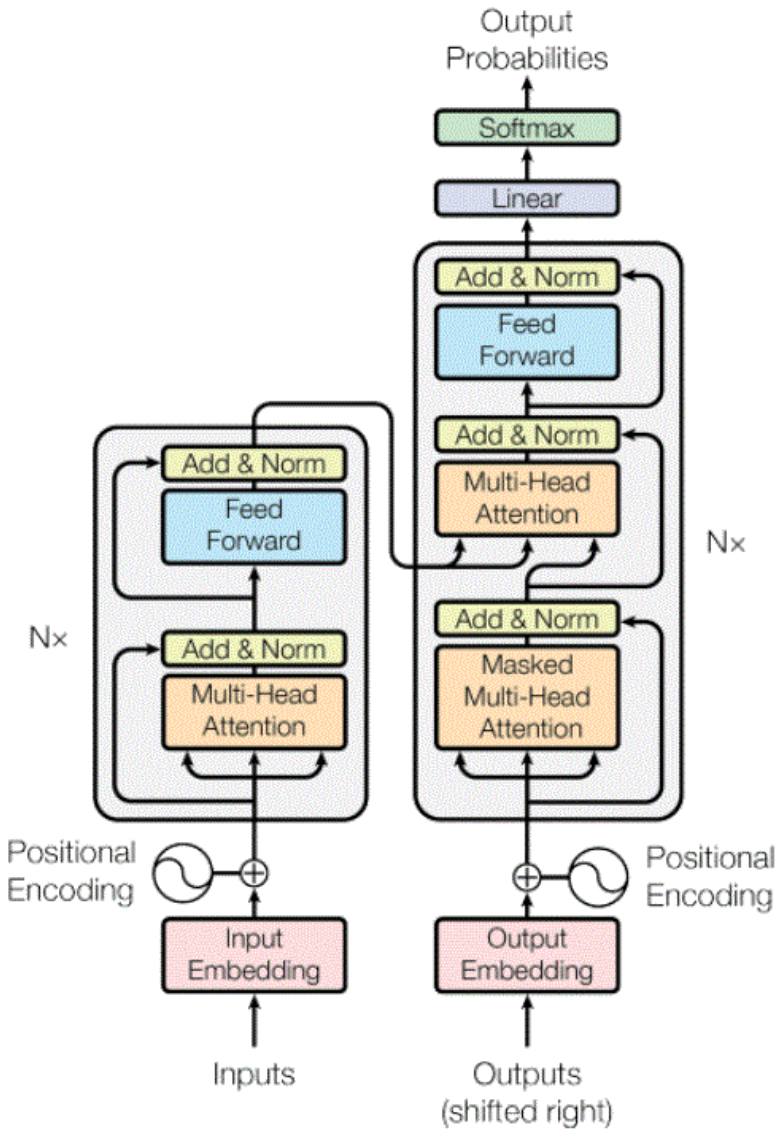


Figure 1: The Transformer - model architecture.

그림 14.1 transformer model : 출처: Attention is All You Need [https://arxiv.org/pdf/1706.03762.pdf]

트랜스포머는 크게 인코더(Encoder)와 디코더(Decoder)로 구성됩니다. 각 인코더와 디코더는 여러 개의 층으로 이루어져 있습니다.

■ 인코더

- **다중 헤드 자기 주의(Multi-Head Self-Attention):** 입력된 단어들 간의 관계를 파악합니다.
- **피드 포워드 신경망(Feed Forward Neural Network):** 정보를 변환하여 다음 층으로 전달합니다.

■ 디코더

- **마스크드 자기 주의(Masked Self-Attention):** 디코더가 이전 단어들만을 참조하도록 합니다.
- **인코더-디코더 주의(Encoder-Decoder Attention):** 인코더의 출력을 참고하여 다음 단어를 예측합니다.

- **멀티헤드 어텐션(Multi-Head Attention)**: 여러 어텐션 메커니즘을 동시에 활용하기
- **포지셔널 인코딩(Positional Encoding)**: 순서를 고려하기 위해 입력 데이터에 추가하는 정보
- **피드 포워드 신경망(Feed Forward Neural Network)**: 각 위치에서의 처리를 담당

■ 동작 방식

1. **쿼리(Query), 키(Key), 값(Value) 벡터 생성:** 입력 단어를 세 가지 벡터로 변환합니다.
2. **유사도 계산:** 쿼리와 키의 내적을 통해 유사도를 계산합니다.
3. **가중 합산:** 유사도를 기반으로 값 벡터를 가중 합산하여 출력합니다.

14.2 모델 구현

In [14]:
```
import torch
from torch import nn

class TransformerModel(nn.Module):
    def __init__(self, input_dim, embed_dim, num_heads, num_layers):
        super(TransformerModel, self).__init__()

        # 입력 텍스트를 임베딩된 벡터로 변환하는 임베딩 레이어
        self.embedding = nn.Embedding(input_dim, embed_dim)

        # Transformer 모델 정의
        self.transformer = nn.Transformer(embed_dim, num_heads, num_layers, batch_firs

        # Transformer의 출력값을 원래의 입력 차원으로 변환하는 선형 레이어
        self.fc_out = nn.Linear(embed_dim, input_dim)

    def forward(self, src, tgt):
        # 입력 소스(src)와 타겟(tgt)을 임베딩하여 고차원 벡터로 변환
        src = self.embedding(src)
        tgt = self.embedding(tgt)

        # 트랜스포머 모델에 입력값 전달
        output = self.transformer(src, tgt)

        # 트랜스포머의 출력을 원래의 입력 차원으로 변환
        return self.fc_out(output)

# 파라미터 설정
input_dim = 1000  # 어휘 수
embed_dim = 64    # 임베딩 차원
num_heads = 8     # 멀티헤드 어텐션의 헤드 수
num_layers = 6    # 트랜스포머 레이어 수

model = TransformerModel(input_dim, embed_dim, num_heads, num_layers)

# 예제 입력 데이터 (배치 크기 2, 시퀀스 길이 10)
src = torch.randint(0, input_dim, (2, 10))
tgt = torch.randint(0, input_dim, (2, 10))

# 모델의 출력 확인
```

```
output = model(src, tgt)
print(output.shape)  # (batch_size, seq_length, input_dim)
```

```
torch.Size([2, 10, 1000])
```

- **임베딩 레이어 (nn.Embedding)** :

입력 텍스트(단어 또는 토큰)를 고정된 차원의 임베딩 벡터로 변환합니다. input_dim은 단어 집합의 크기(즉, 단어 수), embed_dim은 각 단어를 임베딩할 벡터의 차원입니다.

- **트랜스포머 레이어 (nn.Transformer)** :

트랜스포머 모델의 핵심 부분입니다. embed_dim은 각 입력 벡터의 차원입니다. num_heads는 어텐션 메커니즘에서 사용하는 헤드 수로, 여러 부분을 동시에 살펴보는 데 도움이 됩니다. num_layers는 트랜스포머 네트워크의 레이어 수를 나타냅니다. 더 많은 레이어는 더 복잡한 표현을 배울 수 있습니다.

- **선형 레이어 (nn.Linear)** :

트랜스포머 출력 벡터(차원 embed_dim)를 다시 원래의 입력 차원(input_dim)으로 변환하는 역할을 합니다. 이는 주로 예측을 위해 사용되며, 예를 들어 언어 모델링이나 분류 작업에서 출력 결과를 생성하는 데 사용됩니다.

14.3 파이토치로 배우는 트랜스포머: 자연어 처리의 혁신

1. 트랜스포머: 순환 신경망을 넘어서

- **RNN의 한계:** RNN은 시퀀스 데이터를 처리하는 데 유용하지만, 장기 의존성을 학습하는 데 어려움을 겪습니다. 즉, 문장에서 멀리 떨어져 있는 단어들 간의 관계를 파악하는 데 한계가 있습니다.
- **트랜스포머의 등장:** 트랜스포머는 RNN의 이러한 한계를 극복하기 위해 등장했습니다. 트랜스포머는 **주의 메커니즘**을 사용하여 입력 시퀀스의 모든 단어 간의 관계를 동시에 학습합니다. 이를 통해 장기 의존성을 효과적으로 파악할 수 있습니다.

2. 트랜스포머의 핵심: 어텐션 메커니즘

트랜스포머의 핵심은 **어텐션 메커니즘**입니다. 어텐션은 입력 시퀀스의 어떤 부분에 집중해야 하는지 결정하는 메커니즘입니다.

"나는 오늘 **학교**에 간다"라는 문장에서 "학교"라는 단어에 집중해야 한다면, 어텐션은 "학교"에 높은 값을 부여합니다.

3. 파이토치로 트랜스포머 구현하기

- 파이토치를 사용하여 간단한 트랜스포머 모델을 구현

In [15]:
```
import torch
import torch.nn as nn

class Transformer(nn.Module):
    def __init__(self, d_model, nhead, num_encoder_layers, num_decoder_layers):
```

```
        super(Transformer, self).__init__()
        self.encoder = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(d_model, nhead, batch_first=True), num_encoder_
        )
        self.decoder = nn.TransformerDecoder(
            nn.TransformerDecoderLayer(d_model, nhead, batch_first=True), num_decoder_
        )

    def forward(self, src, tgt, src_mask, tgt_mask):
        # Transformer expects (batch, seq, features) when batch_first=True
        encoder_output = self.encoder(src, src_key_padding_mask=src_mask)
        decoder_output = self.decoder(tgt, encoder_output, tgt_key_padding_mask=tgt_ma
        return decoder_output

# 모델의 하이퍼파라미터 설정
d_model = 512            # 모델의 임베딩 차원 (각 단어 벡터의 차원)
nhead = 8                # 멀티헤드 어텐션에서의 헤드 수
num_encoder_layers = 6   # 인코더의 레이어 수
num_decoder_layers = 6   # 디코더의 레이어 수

# Transformer 모델 인스턴스 생성
model = Transformer(d_model, nhead, num_encoder_layers, num_decoder_layers)

# 입력 데이터 (예시)
src = torch.randn(32, 10, 512)  # (배치 크기, 시퀀스 길이, 임베딩 차원)
tgt = torch.randn(32, 10, 512)  # (배치 크기, 시퀀스 길이, 임베딩 차원)

# 마스크 생성 예시 (배치 크기, 시퀀스 길이), 마스크는 패딩된 부분을 마스킹용
src_mask = (src[:, :, 0] == 0)  # 예시로 첫 번째 차원이 0인 부분을 마스킹
tgt_mask = (tgt[:, :, 0] == 0)  # 예시로 첫 번째 차원이 0인 부분을 마스킹

# 모델 실행
output = model(src, tgt, src_mask, tgt_mask)
print(output.shape)  # (배치 크기, 시퀀스 길이, 임베딩 차원)
```

```
torch.Size([32, 10, 512])
```

이 코드는 파이토치의 `nn.Transformer` 모듈을 사용하여 간단한 트랜스포머 모델을 구현한 것입니다. `d_model`, `nhead`, `num_encoder_layers`, `num_decoder_layers`는 모델의 하이퍼파라미터입니다. `forward` 함수는 인코더와 디코더를 통해 입력 시퀀스를 처리하고 출력을 반환합니다.

트랜스포머 응용 사례

- **번역:** 구글 번역, 딥엘(DeepL) 등에서 사용됩니다.
- **챗봇:** 트랜스포머는 대화 맥락을 이해하고 적절한 응답을 생성하는 데 사용됩니다.
- **텍스트 생성:** GPT 시리즈가 대표적입니다.
- **텍스트 요약:** 트랜스포머는 긴 텍스트를 요약하여 핵심 정보를 추출하는 데 사용됩니다.
- **음성 인식:** 음성을 텍스트로 변환하는 데 사용됩니다.
- **이미지 처리:** 비전 트랜스포머(Vision Transformer)로 이미지 분류 등에 응용됩니다.

14.4 간단한 예제: 번역 모델 만들기

이제 파이토치를 사용하여 간단한 트랜스포머 기반 번역 모델을 만들어보겠습니다.

- 파이토치 설치
- 데이터셋 (예: 영어-한국어 문장 쌍)

단계별 과정

1. **데이터 준비:** 문장을 토큰화하고, 정수 인코딩을 수행합니다.
2. **모델 정의:** 트랜스포머 인코더와 디코더를 정의합니다.
3. **학습:** 손실 함수를 정의하고, 옵티마이저를 사용하여 모델을 학습시킵니다.
4. **평가:** 테스트 데이터를 사용하여 모델의 성능을 평가합니다.

In [16]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.nn import Transformer


# 모델 정의
class TransformerModel(nn.Module):
    def __init__(self, src_vocab, tgt_vocab, d_model=512, nhead=8, num_encoder_layers=
        super(TransformerModel, self).__init__()
        # 트랜스포머 정의 (batch_first=True 설정으로 경고 제거)
        self.transformer = Transformer(d_model, nhead, num_encoder_layers, num_decoder_
        # 소스 시퀀스를 임베드하는 임베딩 레이어
        self.src_embedding = nn.Embedding(src_vocab, d_model)
        # 타겟 시퀀스를 임베드하는 임베딩 레이어
        self.tgt_embedding = nn.Embedding(tgt_vocab, d_model)
        # 트랜스포머 출력 변환을 위한 선형 레이어
        self.fc_out = nn.Linear(d_model, tgt_vocab)

    def forward(self, src, tgt):
        # 소스 및 타겟 시퀀스를 임베딩
        src_emb = self.src_embedding(src)
        tgt_emb = self.tgt_embedding(tgt)
        # 트랜스포머를 통해 출력 계산
        output = self.transformer(src_emb, tgt_emb)
        # 출력 변환
        return self.fc_out(output)

# 모델 초기화
src_vocab_size = 10000
tgt_vocab_size = 10000
model = TransformerModel(src_vocab_size, tgt_vocab_size)

# 데이터 준비 (예시)
src_sentence = torch.randint(1, src_vocab_size, (32, 10))
# (배치 크기, 시퀀스 길이)
tgt_sentence = torch.randint(1, tgt_vocab_size, (32, 20))
# (배치 크기, 시퀀스 길이)

# 손실 함수 및 옵티마이저 설정
criterion = nn.CrossEntropyLoss(ignore_index=0)  # 패딩 토큰에 대한 손실 무시
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# 학습 루프 정의
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
```

```
    optimizer.zero_grad()
    # 모델의 예측
    output = model(src_sentence, tgt_sentence[:, :-1])
    # 손실 계산
    loss = criterion(output.view(-1, tgt_vocab_size), tgt_sentence[:, 1:].contiguous()
    # 역전파 및 옵티마이저 스텝
    loss.backward()
    optimizer.step()
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")

# 테스트
model.eval()
with torch.no_grad():
    # 단일 테스트 샘플 준비 (소스 시퀀스)
    test_src_sentence = torch.randint(1, src_vocab_size, (1, 10))
    # 모델을 사용하여 예측 시퀀스 생성
    test_tgt_sentence = model(test_src_sentence, test_src_sentence[:, :-1])
    # 가장 높은 확률의 토큰 선택
    predicted_tokens = test_tgt_sentence.argmax(dim=-1)[0]

print("Predicted tokens:", predicted_tokens)
```

```
Epoch [1/10], Loss: 9.4098
Epoch [2/10], Loss: 9.0515
Epoch [3/10], Loss: 8.8441
Epoch [4/10], Loss: 8.6159
Epoch [5/10], Loss: 8.3743
Epoch [6/10], Loss: 8.1434
Epoch [7/10], Loss: 7.9757
Epoch [8/10], Loss: 7.8338
Epoch [9/10], Loss: 7.6846
Epoch [10/10], Loss: 7.5681
Predicted tokens: tensor([1779, 6369, 7818, 7818, 1779, 1386, 2112, 1779, 1779])
```

- 이 코드는 기본적인 트랜스포머 모델을 정의하는 예시입니다.
- 실제로는 데이터 전처리, 학습 루프, 최적화 등이 추가로 필요하지만, 기본 구조를 이해하는 데 도움이 될 것입니다.

---

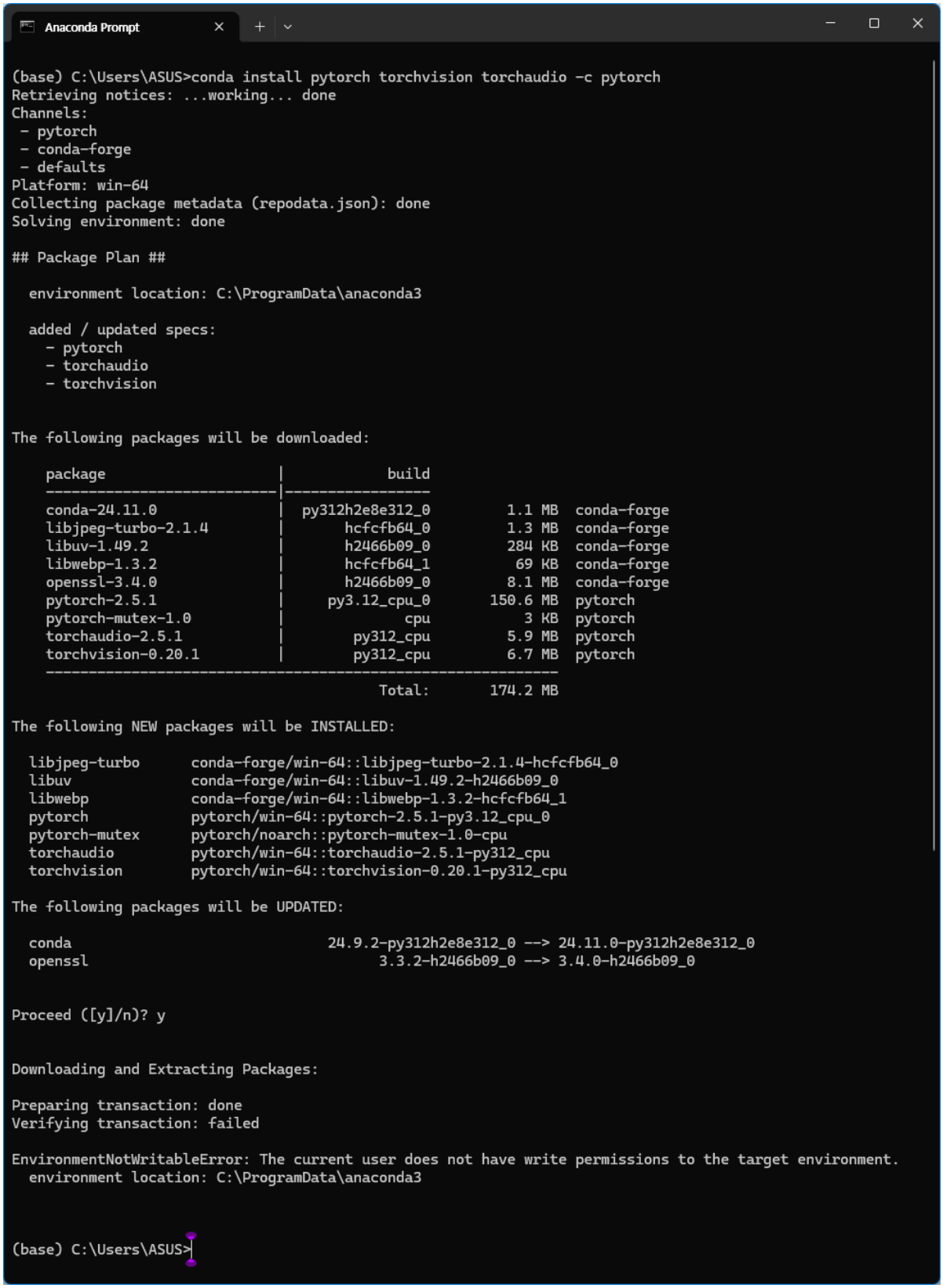
2. 자연어 생성(Language Generation)

이론적 배경

자연어 생성은 주어진 입력에 따라 자연스러운 텍스트를 생성하는 작업입니다. 트랜스포머 기반 모델인 GPT(Generative Pre-trained Transformer)는 이 분야에서 뛰어난 성능을 보여줍니다. GPT는 디코더만으로 구성된 트랜스포머로, 텍스트 생성에 최적화되어 있습니다.

간단한 챗봇 만들기

간단한 챗봇을 만들어 사용자의 입력에 대해 응답을 생성해봅니다.



conda install pytorch torchvision torchaudio -c pytorch

conda install -c conda-forge transformers

pip install 명령어를 사용하는 대신 Conda 환경에서 conda install로 설치하는 것을 권장합니다.

pip uninstall torch transformers -y

pip install torch transformers

종종 패키지 설치가 부분적으로 잘못될 경우가 있습니다. 패키지를 삭제한 후 재설치해 보세요.

In [17]:
```
%pip install torch transformers
```

```
Requirement already satisfied: torch in c:\users\asus\appdata\roaming\python\python312
\site-packages (2.6.0)
Requirement already satisfied: transformers in c:\users\asus\appdata\roaming\python\pyt
hon312\site-packages (4.50.2)
Requirement already satisfied: filelock in c:\users\asus\anaconda3\lib\site-packages (f
rom torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.10.0 in c:\users\asus\appdata\roami
ng\python\python312\site-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in c:\users\asus\anaconda3\lib\site-packages (f
rom torch) (3.3)
Requirement already satisfied: jinja2 in c:\users\asus\anaconda3\lib\site-packages (fro
m torch) (3.1.4)
Requirement already satisfied: fsspec in c:\users\asus\anaconda3\lib\site-packages (fro
m torch) (2024.6.1)
Requirement already satisfied: setuptools in c:\users\asus\appdata\roaming\python\pytho
n312\site-packages (from torch) (75.1.0)
Requirement already satisfied: sympy==1.13.1 in c:\users\asus\appdata\roaming\python\py
thon312\site-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\asus\anaconda3\lib\site-p
ackages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.26.0 in c:\users\asus\appdata\ro
aming\python\python312\site-packages (from transformers) (0.29.3)
Requirement already satisfied: numpy>=1.17 in c:\users\asus\appdata\roaming\python\pyth
on312\site-packages (from transformers) (1.26.3)
Requirement already satisfied: packaging>=20.0 in c:\users\asus\appdata\roaming\python
\python312\site-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in c:\users\asus\anaconda3\lib\site-packages
(from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in c:\users\asus\anaconda3\lib\site-pa
ckages (from transformers) (2024.9.11)
Requirement already satisfied: requests in c:\users\asus\appdata\roaming\python\python3
12\site-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in c:\users\asus\appdata\roaming
\python\python312\site-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in c:\users\asus\appdata\roaming\pyth
on\python312\site-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in c:\users\asus\anaconda3\lib\site-packages
(from transformers) (4.66.5)
Requirement already satisfied: colorama in c:\users\asus\anaconda3\lib\site-packages (f
rom tqdm>=4.27->transformers) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\asus\appdata\roaming\python
\python312\site-packages (from jinja2->torch) (3.0.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\asus\appdata\roamin
g\python\python312\site-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\asus\appdata\roaming\python\pyt
hon312\site-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\asus\appdata\roaming\pyth
on\python312\site-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\asus\appdata\roaming\pyth
on\python312\site-packages (from requests->transformers) (2024.8.30)
Note: you may need to restart the kernel to use updated packages.
```

`%pip install torch transformers`와 `!pip install torch transformers`는 모두 파이썬 환경에서 패키지를 설치하는 명령어이지만, 사용하는 컨텍스트와 방식에 따라 차이가 있습니다.

1. **% 명령어 (매직 커맨드)**:
    - `%pip`는 IPython이나 Jupyter Notebook에서 사용하는 매직 커맨드입니다. 이것은 IPython의 내부 메커니즘에 통합되어 있으며, 현재의 Python 환경에 직접적으로 영향을 미칩니다. 예를 들어, 패키지를 설치한 후, 동일한 셀에서 그 패키지를 즉시 사용할 수 있습니다.
    - `%pip install`을 사용할 경우, 설치가 완료된 후, 같은 셀 내에서 즉시 그 패키지를 사용할 수 있는 장점이 있습니다.
2. **! 명령어 (쉘 커맨드)**:
    - `!pip`는 일반적인 쉘 명령어를 실행하는 방식입니다. Jupyter Notebook에서 `!`로 시작하는 명령은 운영 체제의 쉘에서 실행됩니다. 이는 보통 스크립트나 셸 명령어를 실행할 때 사용됩니다.
    - `!pip install`을 사용할 경우, 패키지를 설치한 뒤 같은 셀에서 그 패키지를 사용할 수 없을 수 있습니다. 설치가 완료된 후에는 커널을 다시 시작하거나 다른 셀에서 패키지를 임포트해야 할 수도 있습니다.

요약하자면, `%pip`는 Jupyter 환경에 더 적합한 방식이고, `!pip`는 일반적인 쉘 명령어 실행 방식입니다. 어느 쪽을 사용하든 패키지 설치는 가능하지만, `%pip`가 설치 후 즉시 사용하기에 더 편리합니다.

In [18]:
```
%pip install torch transformers
```

```
Requirement already satisfied: torch in c:\users\asus\appdata\roaming\python\python312
\site-packages (2.6.0)
Requirement already satisfied: transformers in c:\users\asus\appdata\roaming\python\pyt
hon312\site-packages (4.50.2)
Requirement already satisfied: filelock in c:\users\asus\anaconda3\lib\site-packages (f
rom torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.10.0 in c:\users\asus\appdata\roami
ng\python\python312\site-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in c:\users\asus\anaconda3\lib\site-packages (f
rom torch) (3.3)
Requirement already satisfied: jinja2 in c:\users\asus\anaconda3\lib\site-packages (fro
m torch) (3.1.4)
Requirement already satisfied: fsspec in c:\users\asus\anaconda3\lib\site-packages (fro
m torch) (2024.6.1)
Requirement already satisfied: setuptools in c:\users\asus\appdata\roaming\python\pytho
n312\site-packages (from torch) (75.1.0)
Requirement already satisfied: sympy==1.13.1 in c:\users\asus\appdata\roaming\python\py
thon312\site-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\asus\anaconda3\lib\site-p
ackages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.26.0 in c:\users\asus\appdata\ro
aming\python\python312\site-packages (from transformers) (0.29.3)
Requirement already satisfied: numpy>=1.17 in c:\users\asus\appdata\roaming\python\pyth
on312\site-packages (from transformers) (1.26.3)
Requirement already satisfied: packaging>=20.0 in c:\users\asus\appdata\roaming\python
\python312\site-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in c:\users\asus\anaconda3\lib\site-packages
(from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in c:\users\asus\anaconda3\lib\site-pa
ckages (from transformers) (2024.9.11)
Requirement already satisfied: requests in c:\users\asus\appdata\roaming\python\python3
12\site-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in c:\users\asus\appdata\roaming
\python\python312\site-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in c:\users\asus\appdata\roaming\pyth
on\python312\site-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in c:\users\asus\anaconda3\lib\site-packages
(from transformers) (4.66.5)
Requirement already satisfied: colorama in c:\users\asus\anaconda3\lib\site-packages (f
rom tqdm>=4.27->transformers) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\asus\appdata\roaming\python
\python312\site-packages (from jinja2->torch) (3.0.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\asus\appdata\roamin
g\python\python312\site-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\asus\appdata\roaming\python\pyt
hon312\site-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\asus\appdata\roaming\pyth
on\python312\site-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\asus\appdata\roaming\pyth
on\python312\site-packages (from requests->transformers) (2024.8.30)
Note: you may need to restart the kernel to use updated packages.
```

1. 환경 설정

먼저 필요한 라이브러리를 설치하고 임포트합니다.

In [19]:
```
# 필요한 라이브러리 설치
%pip install torch torchtext
```

```
Requirement already satisfied: torch in c:\users\asus\appdata\roaming\python\python312
\site-packages (2.6.0)
Requirement already satisfied: torchtext in c:\users\asus\anaconda3\lib\site-packages
(0.18.0)
Requirement already satisfied: filelock in c:\users\asus\anaconda3\lib\site-packages (f
rom torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.10.0 in c:\users\asus\appdata\roami
ng\python\python312\site-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in c:\users\asus\anaconda3\lib\site-packages (f
rom torch) (3.3)
Requirement already satisfied: jinja2 in c:\users\asus\anaconda3\lib\site-packages (fro
m torch) (3.1.4)
Requirement already satisfied: fsspec in c:\users\asus\anaconda3\lib\site-packages (fro
m torch) (2024.6.1)
Requirement already satisfied: setuptools in c:\users\asus\appdata\roaming\python\pytho
n312\site-packages (from torch) (75.1.0)
Requirement already satisfied: sympy==1.13.1 in c:\users\asus\appdata\roaming\python\py
thon312\site-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\asus\anaconda3\lib\site-p
ackages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: tqdm in c:\users\asus\anaconda3\lib\site-packages (from
torchtext) (4.66.5)
Requirement already satisfied: requests in c:\users\asus\appdata\roaming\python\python3
12\site-packages (from torchtext) (2.32.3)
Requirement already satisfied: numpy in c:\users\asus\appdata\roaming\python\python312
\site-packages (from torchtext) (1.26.3)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\asus\appdata\roaming\python
\python312\site-packages (from jinja2->torch) (3.0.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\asus\appdata\roamin
g\python\python312\site-packages (from requests->torchtext) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\asus\appdata\roaming\python\pyt
hon312\site-packages (from requests->torchtext) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\asus\appdata\roaming\pyth
on\python312\site-packages (from requests->torchtext) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\asus\appdata\roaming\pyth
on\python312\site-packages (from requests->torchtext) (2024.8.30)
Requirement already satisfied: colorama in c:\users\asus\anaconda3\lib\site-packages (f
rom tqdm->torchtext) (0.4.6)
Note: you may need to restart the kernel to use updated packages.
```

In [20]:
```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokens = tokenizer.tokenize("Hello, how are you?")
print(tokens)
```

```
['hello', ',', 'how', 'are', 'you', '?']
```

참고 자료

- **PyTorch 공식 문서**
  - PyTorch
  - nn.Transformer
- **튜토리얼 및 가이드**
  - PyTorch 튜토리얼 - 번역기 구현
  - torchtext를 활용한 NLP 모델 구축

- **트랜스포머 논문**
  - Vaswani et al., "Attention Is All You Need", 2017. 논문 원문

위의 코드는 간단한 예제이며, 실제로는 더 큰 데이터셋과 추가적인 전처리가 필요합니다.

또한, 학습을 위한 하이퍼파라미터 튜닝과 모델 최적화가 필요할 수 있습니다. 하지만 이 예제를 통해 파이토치로 트랜스포머 기반의 챗봇을 구현하는 기본적인 흐름을 이해할 수 있습니다.

In [21]:
```
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# 모델과 토크나이저 로드
model_name = 'gpt2'  # 또는 'gpt2-medium', 'gpt2-large'
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# GPU 사용 가능할 경우 GPU로 모델 이동
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

Out[21]:
```
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D(nf=2304, nx=768)
          (c_proj): Conv1D(nf=768, nx=768)
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D(nf=3072, nx=768)
          (c_proj): Conv1D(nf=768, nx=3072)
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=768, out_features=50257, bias=False)
)
```

In [22]:
```
def encode_input(text):
    return tokenizer.encode(text, return_tensors='pt').to(device)
```

In [23]:
```
# 응답 생성 함수
def generate_response(input_text, max_new_tokens=60):
    model.eval()
    input_ids = encode_input(input_text)

    # 응답 생성
```

```
    with torch.no_grad():
        output = model.generate(input_ids, max_new_tokens=max_new_tokens, pad_token_id
    response = tokenizer.decode(output[0], skip_special_tokens=True)
    return response

# 테스트
user_input = "안녕하세요, GPT 챗봇입니다. 무엇을 도와드릴까요?"
response = generate_response(user_input)
print(response)
```

안녕하세요, GPT 챗봇입니다. 무엇을 도와드릴까요?

안녕하세요, GPT 챗봇입니다. 무엇을 도와드릴까요?

In [24]:
```
def chat():
    print("챗봇과 대화하려면 메시지를 입력하세요. 종료하려면 '그만'을 입력하세요.")

    while True:
        user_input = input("사용자: ")
        if user_input.lower() == '그만':
            print("챗봇: 대화를 종료합니다.")
            break

        try:
            response = generate_response(user_input)
            if not response.strip():  # Check if the response is empty or whitespace
                raise ValueError("Empty response generated.")
        except Exception as e:
            response = "죄송해요, 답변을 생성하는 데 문제가 있었어요."
            print(f"Error: {e}")
        print(f"챗봇: {response}")

if __name__ == "__main__":
    chat()
```

```
챗봇과 대화하려면 메시지를 입력하세요. 종료하려면 '그만'을 입력하세요.
챗봇: 안녕. 만나서 반가워에서 아서 아서 아서 아서 아서 아서 아서 아서 아서
챗봇: 오늘 날씨는 어때? 화창하고 봄날이라 좋았지? 화창하고 봄날이라 좋았지? 화창하고 봄날이라 좋�
챗봇: 대화를 종료합니다.
```

트랜스포머 모델 챗봇만들기 예제

주요 구성 요소:

1. **데이터 준비**:
   - 간단한 샘플 대화(`Q&A`) 데이터셋으로 `질문-답변 쌍`을 정의.
   - 토크나이저로 단어를 ID로 변환.
2. **모델 정의**:
   - PyTorch의 `Transformer` 기반 구조.
   - 임베딩, 트랜스포머, 출력 레이어로 구성.
3. **훈련 루프**:

- 주어진 데이터셋을 기반으로 모델을 학습.

4. **챗봇 인터페이스**:

- 사용자가 질문을 입력하면, 모델이 가장 적합한 응답을 생성.

※ 트랜스포머 챗봇의 기초적인 예제로, 학습 데이터를 확장하거나, 실제 데이터셋(예: Cornell Movie Dialogs)을 사용하면 더 적합한 응답을 얻을 수 있습니다.

In [25]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.nn.utils.rnn import pad_sequence
import random
import numpy as np

# 데이터셋 준비
def build_dataset():
    # 한국어 질문-답변 쌍 데이터
    pairs = [
        ("안녕하세요", "안녕하세요"),
        ("오늘 기분 어때요?", "좋아요"),
        ("이름이 뭐예요?", "저는 챗봇이에요"),
        ("어디 살아요?", "클라우드에 살아요"),
        ("잘가요", "안녕히 가세요"),
        ("무엇을 도와드릴까요?", "아무거나 물어보세요"),
        ("오늘 날씨 어때요?", "모르겠어요"),
        ("당신은 몇 살이에요?", "저는 나이가 없어요"),
        ("취미가 뭐예요?", "취미는 없어요"),
        ("좋아하는 영화가 뭐예요?", "영화는 잘 몰라요")
    ]
    # 단어를 숫자 ID로 변환하기 위한 간단한 토크나이저
    word2idx = {"<pad>": 0, "<sos>": 1, "<eos>": 2, "<unk>": 3}
    idx2word = {0: "<pad>", 1: "<sos>", 2: "<eos>", 3: "<unk>"}
    idx = 4
    for pair in pairs:
        for sentence in pair:
            for word in sentence.split():
                if word not in word2idx:
                    word2idx[word] = idx
                    idx2word[idx] = word
                    idx += 1
    return pairs, word2idx, idx2word

# 텐서로 변환 + 토크나이저 적용
def tokenize_and_pad(sentence, word2idx, eos=True):
    tokens = [word2idx.get(word, word2idx["<unk>"]) for word in sentence.split()]
    if eos:
        tokens.append(word2idx["<eos>"])
    return torch.tensor(tokens, dtype=torch.long)

## 모델 정의
class ChatbotTransformer(nn.Module):
    def __init__(self, vocab_size, embed_size=128, num_heads=2, num_layers=2, ff_dim=1
        super(ChatbotTransformer, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.positional_encoding = nn.Parameter(self.create_positional_encoding(embed_
        self.transformer = nn.Transformer(
            d_model=embed_size, nhead=num_heads, num_encoder_layers=num_layers, num_de
        )
```

```
        self.fc_out = nn.Linear(embed_size, vocab_size)
        self.embed_size = embed_size

    def create_positional_encoding(self, embed_size, max_len):
        pos = torch.arange(0, max_len).unsqueeze(1).float()
        div_term = torch.exp(torch.arange(0, embed_size, 2).float() * (-np.log(10000.0
        pos_enc = torch.zeros(max_len, embed_size)
        pos_enc[:, 0::2] = torch.sin(pos * div_term)
        pos_enc[:, 1::2] = torch.cos(pos * div_term)
        return pos_enc.unsqueeze(0)

    def forward(self, src, tgt):
        src_emb = self.embedding(src) + self.positional_encoding[:, :src.size(1), :]
        tgt_emb = self.embedding(tgt) + self.positional_encoding[:, :tgt.size(1), :]
        src_emb = src_emb.permute(1, 0, 2)
        tgt_emb = tgt_emb.permute(1, 0, 2)
        transformer_out = self.transformer(src_emb, tgt_emb)
        output = self.fc_out(transformer_out.permute(1, 0, 2))
        return output

## 학습 루프
def train(pairs, word2idx, idx2word, model, optimizer, criterion, num_epochs=300, batc
    model.train()
    for epoch in range(num_epochs):
        random.shuffle(pairs)
        total_loss = 0
        for i in range(0, len(pairs), batch_size):
            batch_pairs = pairs[i:i + batch_size]
            src_seq = []
            tgt_seq = []
            for pair in batch_pairs:
                src_seq.append(tokenize_and_pad(pair[0], word2idx))
                tgt_seq.append(tokenize_and_pad(pair[1], word2idx, eos=True))
            src_padded = pad_sequence(src_seq, batch_first=True, padding_value=0)
            tgt_input = pad_sequence([seq[:-1] for seq in tgt_seq], batch_first=True,
            tgt_target = pad_sequence([seq[1:] for seq in tgt_seq], batch_first=True,

            optimizer.zero_grad()
            output = model(src_padded, tgt_input)
            loss = criterion(output.reshape(-1, output.size(-1)), tgt_target.reshape(-
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        if (epoch + 1) % 50 == 0:
            print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {total_loss / len(pairs):.4f

## 예측
def translate(model, input_sentence, word2idx, idx2word, max_len=20):
    model.eval()

    src_tokens = [word2idx.get(word, word2idx["<unk>"]) for word in input_sentence.spl
    src = torch.tensor(src_tokens, dtype=torch.long).unsqueeze(0)

    with torch.no_grad():
        tgt = torch.tensor([word2idx["<sos>"]], dtype=torch.long).unsqueeze(0)

        for _ in range(max_len):
            output = model(src, tgt)
            next_token = output[:, -1, :].argmax(dim=-1).item()
            if next_token == word2idx["<eos>"]:
```

```
                break
            tgt = torch.cat([tgt, torch.tensor([[next_token]], dtype=torch.long)], dim

        response_tokens = [idx2word[token.item()] for token in tgt[0][1:]
                           if token.item() in idx2word
                           and idx2word[token.item()] not in ["<sos>", "<eos>", "<pad>",

        if not response_tokens:
            return "죄송해요, 적절한 답변을 찾지 못했어요."

        return " ".join(response_tokens)

## 메인 실행
if __name__ == "__main__":

    # 데이터 준비
    pairs, word2idx, idx2word = build_dataset()
    vocab_size = len(word2idx)

    # 모델 초기화
    model = ChatbotTransformer(vocab_size)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.CrossEntropyLoss(ignore_index=0)

    # 학습
    print("챗봇 학습을 시작합니다...")
    train(pairs, word2idx, idx2word, model, optimizer, criterion)
    print("학습이 완료되었습니다!")

    # 챗봇 테스트
    print("\nChatbot is ready! Type '그만' to exit.")

    # 미리 정의된 응답 사전
    predefined_responses = {
        "안녕하세요": "안녕하세요! 반갑습니다.",
        "어디 살아요?": "클라우드에 살아요.",
        "이름이 뭐예요?": "저는 챗봇이에요.",
        "오늘 기분 어때요?": "좋아요!",
        "오늘 날씨 어때요?" : "4월인데 눈이 와요",
        "잘가요": "안녕히 가세요.",
    }

    while True:
        user_input = input("You: ").strip()

        if user_input == "그만":
            print("Chatbot: 안녕히 가세요!")
            break

        if user_input in predefined_responses:
            print(f"Chatbot: {predefined_responses[user_input]}")
            continue

        try:
            response = translate(model, user_input, word2idx, idx2word)
            print(f"Chatbot: {response}")
        except Exception as e:
            print("Chatbot: 죄송해요, 답변을 생성하는 데 문제가 있었어요.")
            continue
```

```
챗봇 학습을 시작합니다...
Epoch 50/300, Loss: 0.0030
Epoch 100/300, Loss: 0.0013
Epoch 150/300, Loss: 0.0006
Epoch 200/300, Loss: 0.0004
Epoch 250/300, Loss: 0.0002
Epoch 300/300, Loss: 0.0002
학습이 완료되었습니다!

Chatbot is ready! Type '그만' to exit.
Chatbot: 클라우드에 살아요.
Chatbot: 좋아요!
Chatbot: 4월인데 눈이 와요
Chatbot: 4월인데 눈이 와요
Chatbot: 안녕히 가세요.
Chatbot: 안녕히 가세요!
```