딥러닝 텍스트 모델 (Deep Learning Text Models)

- 텍스트 데이터의 복잡한 패턴과 문맥적 의미를 학습하는 데 탁월한 성능을 보임
- 순환 신경망(RNN) 계열과 트랜스포머(Transformer) 기반 모델들은 텍스트의 순차적인 특성을 효과적으로 처리

적용 가능한 상황

- 텍스트 분류, 감성 분석, 개체명 인식, 기계 번역, 챗봇, 텍스트 생성 등 복잡한 NLP 문제.
- 대량의 텍스트 데이터에서 미묘한 문맥적 의미와 패턴을 학습해야 할 때.

주의사항

- **데이터 전처리**: 딥러닝 모델은 텍스트를 숫자 벡터(단어 임베딩)로 변환하는 과정이 필수적입니다.
- **계산 자원**: 대규모 딥러닝 모델은 학습에 GPU와 같은 고성능 컴퓨팅 자원을 요구합니다.
- **과적합**: 모델의 복잡도가 높으므로 과적합을 방지하기 위한 규제(Dropout, L1/L2 규제) 및 조기 종료(Early Stopping) 기법이 중요합니다.

1. **RNN (Recurrent Neural Network, 순환 신경망)**:
    - **개념**: 시퀀스 데이터(텍스트, 시계열 등) 처리에 특화된 신경망으로, 이전 단계의 정보를 현재 단계의 계산에 재활용하는 '순환(recurrent)' 구조를 가집니다. 이를 통해 단어의 순서와 문맥을 학습할 수 있습니다.
    - **단점**: 장기 의존성(long-term dependencies) 문제, 즉 시퀀스가 길어질수록 초반 정보가 손실되는 문제가 있습니다.
2. **LSTM (Long Short-Term Memory, 장단기 기억 신경망)**:
    - **개념**: RNN의 장기 의존성 문제를 해결하기 위해 고안된 특별한 종류의 RNN입니다. '게이트(gate)' 메커니즘(입력 게이트, 망각 게이트, 출력 게이트)을 통해 정보를 선택적으로 기억하거나 잊어버리면서 장기적인 문맥 정보를 효과적으로 유지할 수 있습니다.
    - **GRU (Gated Recurrent Unit)**: LSTM의 간소화된 버전으로, 게이트 수가 적어 학습 속도가 빠르면서도 LSTM과 유사한 성능을 보입니다.
3. **CNN (Convolutional Neural Network, 합성곱 신경망)**:
    - **개념**: 주로 이미지 처리에 사용되지만, 텍스트에서도 지역적인 특징(n-gram과 유사)을 추출하는 데 활용될 수 있습니다. 필터(filter)를 사용하여 텍스트의 특정 패턴을 감지합니다.
    - **적용**: 텍스트 분류, 감성 분석 등에서 RNN/LSTM과 함께 또는 단독으로 사용됩니다.
4. **Transformer (트랜스포머)**:
    - **개념**: RNN의 순환 구조를 제거하고 **어텐션(Attention) 메커니즘**만을 사용하여 시퀀스 데이터를 처리하는 모델입니다. 특히 '셀프 어텐션(Self-Attention)'을 통해 시퀀스 내의 모든 단어 쌍 간의 관계를 동시에 고려하여 문맥을 파악합니다.
    - **장점**: 병렬 처리가 가능하여 학습 속도가 빠르고, 장기 의존성 문제를 효과적으로 해결합니다.
    - **적용**: BERT, GPT 등 최신 대규모 언어 모델의 기반 구조입니다.

`PyTorch` 및 `Keras/TensorFlow` 활용

두 프레임워크 모두 딥러닝 텍스트 모델을 구축하는 데 널리 사용됩니다.

- `PyTorch`:
  - **특징**: 동적 계산 그래프(Dynamic Computation Graph)를 사용하여 디버깅이 용이하고 유연성이 높습니다. 연구 및 개발 분야에서 선호됩니다.
  - **주요 모듈**: `torch.nn` (신경망 레이어), `torch.optim` (옵티마이저), `torch.utils.data` (데이터 로더).
  - **텍스트 처리**: `torchtext` 라이브러리를 통해 텍스트 전처리 및 데이터 로딩을 쉽게 할 수 있습니다.

- `Keras/TensorFlow`:
  - **특징**: 정적 계산 그래프(Static Computation Graph)를 사용하며, Keras는 TensorFlow 위에서 동작하는 고수준 API로, 사용하기 쉽고 빠르게 모델을 구축할 수 있습니다. 프로덕션 환경에서 널리 사용됩니다.
  - **주요 모듈**: `tf.keras.layers` (신경망 레이어), `tf.keras.optimizers` (옵티마이저), `tf.keras.preprocessing.text` (텍스트 전처리).

코드 예시 (Keras를 이용한 간단한 LSTM 텍스트 분류)

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, SpatialDropout1D
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# 1. 텍스트 데이터 준비 (예시)
texts = [
    "I love this movie, it's amazing!",
    "This film was terrible, a complete waste of time.",
    "The acting was good but the plot was boring.",
    "Highly recommend this, a must-watch.",
    "Never watch this again, so bad.",
    "It was okay, not great not terrible."
]
labels = np.array([1, 0, 0, 1, 0, 0]) # 1: 긍정, 0: 부정

# 2. 텍스트 전처리 및 벡터화 (토큰화 및 시퀀스 패딩)
# num_words: 사용할 단어의 최대 개수
tokenizer = Tokenizer(num_words=1000, oov_token="<unk>")
tokenizer.fit_on_texts(texts)
word_index = tokenizer.word_index

# 텍스트를 시퀀스로 변환
sequences = tokenizer.texts_to_sequences(texts)

# 시퀀스 패딩 (길이 맞추기)
# maxlen: 시퀀스의 최대 길이
```

```
# padding: 'pre' (앞에 0 채우기) 또는 'post' (뒤에 0 채우기)
padded_sequences = pad_sequences(sequences, maxlen=10, padding='post')

# 3. 훈련/테스트 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels,
test_size=0.3, random_state=42, stratify=labels)

# 4. LSTM 모델 구축
# vocab_size: 단어 사전의 크기
# embedding_dim: 임베딩 벡터의 차원
# input_length: 입력 시퀀스의 길이 (maxlen과 동일)
model = Sequential([
    Embedding(input_dim=len(word_index) + 1, output_dim=100, input_length=10),
    SpatialDropout1D(0.2), # 과적합 방지
    LSTM(100, dropout=0.2, recurrent_dropout=0.2),
    Dense(1, activation='sigmoid') # 이진 분류이므로 sigmoid
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
'''

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 10, 100)           3400
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 10, 100)           0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               80400
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 83,901
Trainable params: 83,901
Non-trainable params: 0
'''

# 5. 모델 학습
history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.2, verbose=0)

# 6. 예측 및 평가
y_pred_proba = model.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)

print("\n--- LSTM 텍스트 분류 모델 평가 ---")
print(classification_report(y_test, y_pred))
'''
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       0.00      0.00      0.00         1

    accuracy                           0.50         2
```

```
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
'''
```

결과 해석 방법

- 딥러닝 모델은 내부적으로 복잡한 비선형 변환을 수행하므로, 모델의 각 계수나 레이어의 의미를 직접적으로 해석하기는 어렵습니다.
- 모델의 성능은 정확도, 정밀도, 재현율, F1-점수, AUC 등 일반적인 분류 평가지표를 통해 평가합니다.
- `model.summary()`를 통해 모델의 레이어 구성과 파라미터 수를 확인할 수 있습니다.
- `history` 객체를 통해 학습 과정에서의 손실(loss)과 정확도(accuracy) 변화를 시각화하여 과적합 여부를 판단할 수 있습니다.

장단점 및 대안

- **장점**:
  - 텍스트의 복잡한 문맥적 의미와 장기 의존성을 효과적으로 학습하여 매우 높은 성능을 달성할 수 있습니다.
  - 대량의 데이터가 주어졌을 때, 기존 머신러닝 모델의 성능을 뛰어넘는 경우가 많습니다.
- **단점**:
  - 대량의 데이터와 높은 컴퓨팅 자원(GPU)이 필요합니다.
  - 모델의 구조가 복잡하여 학습 시간이 길고, 하이퍼파라미터 튜닝이 까다롭습니다.
  - 모델의 내부 동작을 해석하기 매우 어렵습니다.
- **대안**:
  - **사전 학습된 언어 모델 (Pre-trained Language Models)**: BERT, GPT, RoBERTa 등 대규모 텍스트 코퍼스로 미리 학습된 모델을 파인튜닝(fine-tuning)하여 사용하면, 적은 데이터로도 매우 높은 성능을 달성할 수 있습니다.
  - **전통적인 머신러닝 모델**: 데이터의 양이 적거나, 모델의 해석이 중요할 때는 로지스틱 회귀, SVM, 나이브 베이즈 등 전통적인 모델이 더 적합할 수 있습니다.