□ 강의 개요

학습 목표:

- 이 강의의 핵심 개념을 이해합니다
- 실전에 적용할 수 있는 지식을 습득합니다

주요 키워드: [자동으로 채워질 예정]

선행 지식: 기본적인 컴퓨터 사용 능력

Lecture 6: 문자 임베딩과 오토인코더

Character Embeddings and Autoencoders

CSCI E-89B Introduction to Natural Language Processing

October 26, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 06

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 06의 핵심 개념 학습

□ 핵심 요약

이번 강의에서는 단어를 구성하는 개별 문자를 벡터로 표현하는 **문자 임베딩**을 학습합니다. 이를 통해 OOV(Out-of-Vocabulary) 문제와 철자 오류에 강건한 모델을 만들 수 있습니다. 다음으로, 레이블 없는 데이터로부터 효율적인 데이터 표현을 학습하는 비지도 학습 모델인 **오토인코더**를 다룹니다.
오토인코더의 기본 구조, 스택 오토인코더, 희소 오토인코더, 그리고 변형 오토인코더의 개념과 활용법을 알아봅니다.

Contents

1 용어 정리

[title=핵심 용어 정리]

용어	쉬운 설명	원어 / 관련 개념
문자 임베딩	단어가 아닌 개별 문자(character)를 벡터로 변환하는 기술. 접두사, 접미사 등 단어 내부 구조를 학습할 수 있음.	Character Embeddings
오토인코더	데이터를 압축(인코딩)했다가 다시 원본으로 복원(디코딩)하도록 학습하는 신경망. 차원 축소, 특징 추출 등에 사용됨.	Autoencoder (AE)
인코더	오토인코더의 일부로, 입력 데이터를 저차원의 잠재 공간(latent space) 벡터로 압축하는 역할.	Encoder
디코더	오토인코더의 일부로, 압축된 잠재 벡터를 다시 원본 데이터 차원으로 복원하는 역할.	Decoder
재구성 손실	오토인코더의 원본 입력과 디코더가 복원한 출력 간의 차이. 이 손실을 최소화하는 방향으로 학습이 진행됨.	Reconstruction Loss
t-SNE	고차원 데이터를 시각화하기 좋은 2차원 또는 3차원으로 축소하는 알고리즘. 데이터 클러스터를 시각적으로 확인할 때 유용함.	t-Distributed Stochastic Neighbor Embedding
희소 오토인코더	오토인코더의 잠재 공간(코딩) 뉴런 중 일부만 활성화되도록 제약을 가하는 방식. 데이터의 특징을 더 명확하게 분리할 수 있음.	Sparse Autoencoder
KL 발산	두 확률 분포의 차이를 측정하는 지표. 희소 오토인코더에서 목표 활성도(sparsity)와 실제 활성도의 차이를 줄이는 손실 함수로 사용됨.	Kullback-Leibler Divergence
변형 오토인코더	잠재 공간을 정규분포 같은 연속적인 확률 분포로 만드는 생성 모델. 새로운 데이터를 생성하는 데 강점이 있음.	Variational Autoencoder (VAE)

2 문자 임베딩 (Character Embeddings)

2.1 개요

문자 임베딩은 단어(word) 단위가 아닌, 단어를 구성하는 개별 문자(character)를 연속적인 벡터 공간에 표현하는 방법입니다.

이 접근법은 모델이 단어의 철자(orthographic) 및 형태론적(morphological) 특징을 학습하도록 돕습니다. 예를 들어, 'running', 'runner', 'ran'과 같은 단어들이 'run'이라는 어근을 공유한다는 것을 파악할 수 있게 됩니다.

문자 임베딩의 주요 장점

- **Out-of-Vocabulary (OOV) 문제 완화**: 훈련 데이터에 없던 새로운 단어가 등장해도, 그 단어를 구성하는 문자들은 이미 학습했을 가능성이 높으므로 의미 있는 벡터 표현을 생성할 수 있습니다.
- **철자 오류(Misspellings)에 강건함**: 단어에 오타가 있어도, 대부분의 문자는 올바르기 때문에 어느 정도 유사한 벡터 표현을 유지할 수 있습니다.
- **형태론적 정보 포착**: 접두사(prefix), 접미사(suffix), 어근(stem)과 같은 단어 내부의 미세한 구조를 포착하여, 형태론적으로 풍부한 언어(예: 핀란드어, 터키어)에서 특히 효과적입니다.

2.2 적용 분야

- **형태론이 복잡한 언어 처리**: 핀란드어는 15개의 문법적 격(case)을 가지며, 터키어는 어근에 많은 접사가 붙어 단어가 길어집니다. 이런 언어에서는 단어 사전 크기가 매우 커지므로, 문자 임베딩을 통해 모델을 더 효율적으로 만들 수 있습니다.
- **개체명 인식 (Named Entity Recognition, NER)**: 사람 이름, 지명 등은 매우 다양하고 훈련 데이터에 등장하지 않은 경우가 많습니다. 문자 임베딩은 이런 새로운 개체명을 인식하는 데 도움을 줍니다.
- **맞춤법 교정**: 단어의 철자 오류를 탐지하고 교정하는 모델을 자연스럽게 구축할 수 있습니다.

2.3 구현 방법

문자 임베딩은 주로 RNN, CNN과 같은 신경망 모델이나 하이브리드 접근법을 통해 구현됩니다.

- **Character-Level RNNs/CNNs**: 문자의 시퀀스를 RNN이나 CNN에 입력하여 단어 벡터를 동적으로 생성합니다. RNN은 순차적 정보를, CNN은 지역적 패턴(예: 'ing' 접미사)을 포착하는 데 유리합니다.
- **Embedding Layer**: 각 문자를 고유한 정수 인덱스에 매핑한 후, 임베딩 레이어를 통과시켜 저차원의 밀집 벡터(dense vector)로 변환합니다.
- **Hybrid Approaches**: 문자 임베딩과 단어 임베딩을 결합하여 사용하는 방식입니다. 두 임베딩 벡터를 연결(concatenate)하여 모델의 입력으로 사용하면, 단어의 미세 구조와 전체적인

의미를 모두 활용할 수 있습니다.

2.4 Python 실습 예제

TensorFlow/Keras를 사용하여 문자 임베딩을 적용한 간단한 텍스트 분류 모델을 구현하는 예제입니다.

문자 임베딩을 위한 데이터 전처리

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 샘플데이터
texts = ["hello world", "machine learning", "deep learning"]
labels = np.array([1, 0, 0])

# 1. 문자수준토큰화 (Character-level Tokenization)
# 의Tokenizer char_level=True 옵션사용
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(texts)

# 텍스트를정수시퀀스로변환
sequences = tokenizer.texts_to_sequences(texts)
print정수(" 시퀀스:", sequences)

# 문자사전 (Vocabulary)
char_index = tokenizer.word_index
print문자(" 인덱스:", char_index)

# 2. 패딩 (Padding)
# 모든시퀀스의길이를통일하기위해가장긴시퀀스에맞춰을 0 채움
max_sequence_length = max(len(seq) for seq in sequences)
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length,
    padding='post')
print패딩된(" 시퀀스:\n", padded_sequences)
```

Listing 1: Character-level tokenization and padding

사전 크기 및 패딩

`tokenizer.word_index` . (

문자 임베딩을 사용한 LSTM 모델 구축

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# 사전크기은 (+1 패딩용인덱스 0)
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 8 # 임베딩벡터의차원

```

```
# 모델구축
model = Sequential([
    # Embedding Layer: 정수인덱스를  embedding_dim 차원의벡터로변환
    Embedding(input_dim=vocab_size,
              output_dim=embedding_dim,
              input_length=max_sequence_length),
    LSTM(64),
    Dense(1, activation='sigmoid') # 이진분류
])

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

model.summary()

# 모델훈련
model.fit(padded_sequences, labels, epochs=10, batch_size=2)
```

Listing 2: Building and training a model with character embeddings

이 모델에서 임베딩 레이어는 훈련 과정 중에 데이터에 맞게 문자 벡터를 학습합니다. 문자 수가 단어 수보다 훨씬 적기 때문에, 임베딩 차원(embedding$_dim$) (: 8, 16).

3 오토인코더 (Autoencoders)

3.1 기본 개념

오토인코더는 레이블이 없는 데이터로부터 효율적인 데이터 표현(인코딩)을 학습하기 위한 비지도 학습(unsupervised learning) 방식의 인공 신경망입니다.

오토인코더는 두 부분으로 구성됩니다:

- **인코더** (Encoder): 입력 데이터를 저차원의 잠재 공간(latent space)으로 압축합니다. 이 압축된 표현을 **인코딩**(encoding) 또는 **코딩**(coding)이라고 합니다.
- **디코더** (Decoder): 인코딩된 표현을 다시 원래의 입력 데이터와 같은 차원으로 복원합니다.

학습 목표는 입력 데이터와 디코더가 복원한 출력 데이터 간의 차이, 즉 **재구성 손실** (reconstruction loss)을 최소화하는 것입니다.

Figure: 오토인코더의 기본 구조. 입력(x)이 인코더를 통해 저차원 표현(z)으로 압축된 후, 디코더를 통해 다시 원본과 유사한 출력(x')으로 복원됩니다.

오토인코더의 직관적 예시

두 가지 숫자 시퀀스를 기억해야 한다고 가정해봅시다.

- **시퀀스 1**: 5, 7, 3, 5, 8, 9, 67, 5, ... (규칙 없음)
- **시퀀스 2**: 70, 68, 66, 64, 62, ... (규칙: 70에서 시작해서 2씩 감소)

시퀀스 1은 모든 숫자를 개별적으로 외워야 하지만, 시퀀스 2는 "시작 값 70, 규칙 -2"라는 훨씬 적은 정보로 압축하여 기억할 수 있습니다. 오토인코더는 이처럼 데이터 내의 패턴이나 구조를 학습하여 데이터를 효율적으로 압축하고 표현하는 방법을 배웁니다.

3.2 스택 오토인코더 (Stacked Autoencoders)

스택 오토인코더는 여러 개의 은닉층을 쌓아 만든 깊은(deep) 오토인코더입니다. 일반적으로 인코더와 디코더가 대칭적인 구조를 가집니다.

예를 들어, 28x28 픽셀 (총 784개) 크기의 이미지를 압축하는 경우 다음과 같은 구조를 가질 수 있습니다.

$$784 \to 100 \to 30(\text{인코딩}) \to 100 \to 784$$

여기서 30차원 벡터가 이미지의 핵심 특징을 담은 인코딩이 됩니다. 이처럼 내부 표현의 차원 수가 입력보다 낮은 오토인코더를 **불완전**(undercomplete) **오토인코더**라고 하며, 이는 모델이 데이터의 가장 중요한 특징을 학습하도록 강제합니다.

Figure: 스택 오토인코더 예시. 784차원 입력을 30차원의 코딩으로 압축한 후

다시 *784*차원으로 복원합니다.

3.3 활용: 비지도 사전학습 (Unsupervised Pretraining)

오토인코더의 가장 강력한 활용 사례 중 하나는 레이블이 부족한 대규모 데이터셋에서 특징 추출기(feature extractor)를 학습하는 것입니다.

비지도 사전학습 절차

Phase 1: 오토인코더 훈련 레이블 유무에 상관없이 전체 데이터를 사용하여 스택 오토인코더를 훈련시킵니다. 이 과정을 통해 인코더는 데이터를 잘 표현하는 방법을 학습하게 됩니다.

Phase 2: 분류기 훈련 훈련된 오토인코더에서 디코더 부분은 버리고 인코더 부분만 가져옵니다.
이 인코더 위에 새로운 분류기(예: Softmax 층)를 추가합니다.
이제 레이블이 있는 소량의 데이터만을 사용하여 이 새로운 모델(인코더 + 분류기)을 훈련시킵니다. 이때 인코더의 가중치는 고정(freeze)하거나 미세 조정(fine-tuning)할 수 있습니다.

이 방식은 적은 양의 레이블 데이터로도 높은 성능의 분류기를 만들 수 있게 해줍니다. 인코더가 대규모 비지도 데이터로부터 이미 데이터의 유용한 구조를 학습했기 때문입니다.

Figure: 오토인코더를 이용한 비지도 사전학습. *1*단계에서 전체 데이터로 오토인코더를 학습하고, *2*단계에서 학습된 인코더를 재사용하여 레이블된 데이터로 분류기를 학습합니다.

3.4 잠재 공간 시각화: t-SNE

오토인코더가 학습한 잠재 공간(latent space, 즉 인코딩들의 공간)이 데이터를 얼마나 잘 군집화했는지 확인하기 위해 t-SNE와 같은 시각화 도구를 사용합니다.

t-SNE는 고차원의 인코딩 벡터(예: 30차원)를 2차원 평면에 매핑하여, 비슷한 데이터들이 가깝게 모이고 다른 데이터들은 멀리 떨어지도록 배치합니다. 이를 통해 각 클러스터가 어떤 종류의 데이터에 해당하는지(예: 신발, 가방, 셔츠) 시각적으로 확인할 수 있습니다.

t-SNE를 이용한 인코딩 시각화 코드 예시

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# 1. 훈련된인코더를사용하여검증데이터의인코딩압축된     ( 표현)을 얻음
X_valid_compressed = stacked_encoder.predict(X_valid)

# 2. t-SNE 모델을사용하여차원  30 인코딩을차원으로 2 변환
tsne = TSNE()
X_valid_2D = tsne.fit_transform(X_valid_compressed)
```

```
10
11 # 3. 차원2 평면에산점도로시각화색상은    ( 실제레이블 )
12 plt.scatter(X_valid_2D[:, 0], X_valid_2D[:, 1], c=y_valid, s=10, cmap="
       tab10")
13 plt.show()
```

Listing 3: Visualizing codings with t-SNE

4 희소 오토인코더 (Sparse Autoencoders)

4.1 개념과 필요성

기본적인 불완전 오토인코더는 병목(bottleneck) 층의 뉴런 수를 줄여 데이터 압축을 유도합니다. 하지만 만약 데이터가 매우 다양하다면(예: 숫자, 동물, 자동차 이미지가 섞여 있는 경우), 작은 병목 층만으로는 모든 종류의 특징을 학습하기 어려울 수 있습니다.

이때 병목 층의 뉴런 수를 입력보다 크게 설정하는 **과완전(overcomplete) 오토인코더**를 사용할 수 있지만, 이 경우 모델이 단순히 입력을 복사하는 항등 함수(identity function)를 학습할 위험이 있습니다.

희소 오토인코더는 과완전 오토인코더를 사용하면서도, 인코딩 층의 뉴런 대부분이 비활성화(출력이 0에 가까움)되도록 강제하는 규제(regularization)를 추가한 모델입니다. 즉, 각 입력에 대해 소수의 뉴런만 활성화하여 해당 입력의 특정 특징을 전문적으로 감지하도록 유도합니다.

희소 오토인코더의 작동 원리

이미지 데이터셋에 숫자 '7'과 '8'이 있다고 가정합시다. 희소 오토인코더는 다음과 같이 학습할 수 있습니다.

- 어떤 뉴런은 '7'의 위쪽 가로선을 감지할 때만 강하게 활성화됩니다.
- 다른 뉴런은 '8'의 아래쪽 원형 부분을 감지할 때만 강하게 활성화됩니다.
- '7' 이미지가 입력되면 '가로선' 뉴런은 활성화되지만, '아래쪽 원' 뉴런은 비활성화됩니다.

이처럼 각 뉴런이 데이터의 특정 부분 특징을 전담하게 하여, 더 유용하고 해석 가능한 특징을 학습할 수 있습니다.

4.2 구현 방법

희소성을 강제하기 위해 손실 함수에 규제 항을 추가합니다. 대표적인 두 가지 방법이 있습니다.

4.2.1 L1 활동 규제 (L1 Activity Regularization)

인코딩 층의 활성화 값(activation)에 대해 L1 손실을 추가합니다. L1 규제는 가중치를 0으로 만드는 경향이 있으므로, 많은 뉴런의 활성화 값을 0에 가깝게 만들어 희소성을 유도합니다.

$$\text{Total Loss} = \text{Reconstruction Loss} + \lambda \sum_i |a_i|$$

여기서 a_i는 인코딩 층의 i번째 뉴런의 활성화 값이고, λ는 규제 강도를 조절하는 하이퍼파라미터입니다.

Keras에서 L1 활동 규제 적용하기

```
from tensorflow.keras.layers import ActivityRegularization

# ... 인코더모델정의 ...
# 인코딩층 (Dense) 바로뒤에 ActivityRegularization 레이어를추가
keras.layers.Dense(300, activation="sigmoid"),
keras.layers.ActivityRegularization(l1=1e-3)
# ...
```

Listing 4: Applying L1 Activity Regularization

4.2.2 쿨백-라이블러 발산 (Kullback-Leibler Divergence)

더 정교한 방법으로, 각 뉴런의 평균 활성도를 특정 목표값(target sparsity, 예: 0.1)에 가깝게 유지하도록 강제합니다. 이는 KL 발산을 사용하여 구현됩니다.

KL 발산은 두 확률 분포의 차이를 측정하며, 여기서는 "뉴런이 활성화될 실제 확률(평균 활성도)"과 "목표 확률(target sparsity)" 사이의 차이를 측정합니다.

$$D_{KL}(p||\hat{p}) = p \log \frac{p}{\hat{p}} + (1-p) \log \frac{1-p}{1-\hat{p}}$$

여기서 p는 목표 희소성(예: 0.1), $\hat{p}$는 배치(batch) 데이터에 대한 뉴런의 실제 평균 활성도입니다. 이 D_{KL} 값이 손실 함수에 추가되어, $\hat{p}$가 p에 가까워지도록 모델을 훈련시킵니다.

Figure: KL 발산 손실 함수. 실제 활성도(Actual sparsity)가 목표 활성도(Target sparsity)와 일치할 때 비용(Cost)이 0이 되고, 멀어질수록 급격히 증가합니다.

주의사항

KL 발산은 L1 규제보다 더 강하게 목표 희소성을 강제하는 경향이 있습니다. 손실 함수의 모양이 목표 지점에서 더 뾰족하기(steeper) 때문에, 수렴 속도가 더 빠를 수 있습니다.

5 변형 오토인코더 (Variational Autoencoders, VAEs)

5.1 개념과 목적

일반 오토인코더가 학습한 잠재 공간은 불연속적일 수 있습니다. 즉, 두 개의 유효한 인코딩 사이의 중간 지점에 있는 벡터가 의미 없는 출력으로 디코딩될 수 있습니다.

변형 오토인코더(VAE)는 잠재 공간이 잘 구조화되고 연속적이도록 만드는 **생성 모델(generative model)**입니다. VAE는 입력을 하나의 고정된 벡터로 인코딩하는 대신, 평균(μ)과 표준편차(σ)를 갖는 확률 분포(일반적으로 가우시안 분포)로 인코딩합니다.

디코더는 이 분포에서 랜덤하게 샘플링한 벡터를 사용하여 입력을 재구성합니다. 이러한 확률적 접근 방식은 다음과 같은 장점을 가집니다.

- **연속적인 잠재 공간**: 비슷한 입력들은 잠재 공간에서 서로 겹치는 분포로 표현됩니다. 따라서 한 분포의 지점에서 다른 분포의 지점으로 점진적으로 이동하면, 출력도 자연스럽게 변환됩니다 (예: 숫자 '6' 이미지가 '0' 이미지로 부드럽게 변함).
- **새로운 데이터 생성**: 잠재 공간에서 임의의 점을 샘플링하여 디코더에 통과시키면, 훈련 데이터와 유사하지만 완전히 새로운 데이터를 생성할 수 있습니다.

Figure: VAE 아키텍처. 인코더는 평균*(μ)*과 로그 분산*($\log \sigma$)*을 출력하고, 가우시안 노이즈를 더해 샘플링된 코딩을 생성합니다. 이 코딩이 디코더의 입력이 됩니다.

5.2 손실 함수

VAE의 손실 함수는 두 가지 요소로 구성됩니다.

$$\texttt{Total Loss} = \texttt{Reconstruction Loss} + \texttt{Latent Loss}$$

1. 재구성 손실 (Reconstruction Loss) 일반 오토인코더와 동일하게, 원본 입력과 디코더의 출력 간의 차이를 측정합니다.

2. 잠재 손실 (Latent Loss) 인코더가 생성한 분포가 목표 분포(일반적으로 표준 정규 분포, 즉 평균=0, 분산=1)에 가깝도록 강제하는 규제 항입니다. 이 손실은 KL 발산을 사용하여 계산됩니다.

$$\mathcal{L}_{\texttt{latent}} = D_{KL}(\mathcal{N}(\mu, \sigma^2) || \mathcal{N}(0, 1)) = -\frac{1}{2}\sum_{i=1}^{n}(1 + \log \sigma_i^2 - \sigma_i^2 - \mu_i^2)$$

이 손실 항은 모델이 잠재 공간에 분포들을 너무 멀리 흩어놓지 않고, 원점 근처에 잘 군집화하도록 유도합니다. 만약 이 항이 없다면, 인코더는 재구성 손실

을 줄이기 위해 각 분포의 분산(σ)을 0으로 만들어 사실상 일반 오토인코더처럼 작동하려 할 것입니다.

VAE의 핵심 트레이드오프

VAE는 재구성(입력을 잘 복원하는 것)과 규제(잠재 공간을 잘 구조화하는 것) 사이의 균형을 맞추는 과정입니다. 잠재 손실이 너무 강하면 이미지가 흐릿하게 재구성될 수 있고, 재구성 손실만 강조하면 잠재 공간의 연속성이 떨어질 수 있습니다.

6 빠르게 훑어보기 (1-Page Summary)

문자 임베딩 (Character Embeddings)

정의: 단어 대신 개별 문자를 벡터로 표현.
장점: OOV 문제, 철자 오류에 강건. 형태론적 정보 포착.
적용: 형태론이 복잡한 언어(터키어 등), 개체명 인식(NER).
구현: Tokenizer(char$_level = True$), $EmbeddingLayer + RNN/CNN$.

오토인코더 (Autoencoders)

정의: 비지도 학습으로 데이터의 효율적 표현을 학습하는 신경망 (인코더 + 디코더).
목표: 재구성 손실(입력-출력 차이) 최소화.
종류:

- **스택 AE**: 여러 층을 쌓아 깊게 만듦 ($784 \to 100 \to 30 \to 100 \to 784$).
- **희소 AE**: 인코딩 층 뉴런의 일부만 활성화되도록 규제 (L1, KL 발산).
- **변형 AE (VAE)**: 생성 모델. 잠재 공간을 확률 분포로 만들어 연속성을 확보.

핵심 활용: 비지도 사전학습 (레이블 없는 데이터로 인코더 학습 후, 소량의 레이블 데이터로 분류기 학습).

주요 규제 기법

- **L1 활동 규제**:
  - **목표**: 인코딩 층의 활성화 값을 희소하게(sparse) 만듦.
  - **방법**: 손실 함수에 활성화 값의 절댓값 합(L1 norm)을 추가.
- **KL 발산 규제**:
  - **목표 (희소 AE)**: 뉴런의 평균 활성도를 특정 목표값(예: 10%)으로 유도.
  - **목표 (VAE)**: 잠재 분포를 표준 정규 분포에 가깝게 만듦.
  - **방법**: 두 확률 분포(실제 vs 목표)의 차이를 측정하여 손실에 추가.