chapter 4 - word2vec 속도 개선

- CBOW 모델은 말뭉치에 포함된 어휘 수가 많아지면 계산량도 커진다는 문제를 가지고 있습니다.
- 그래서 이번 장의 목표는 word2vec의 속도 개선으로 잡아봤습니다.
- 첫 번째 개선으로는 Embedding이라는 새로운 계층을 도입합니다. 그리고 두 번째 개선으로 네거티브 샘플링이라는 새로운 손실 함수를 도입합니다.

4.1 word2vec 개선

- [그림 4-1]의 CBOW 모델도 작은 말뭉치를 다룰 때는 특별히 문제될 게 없습니다.
- 그러나 거대한 말뭉치를 다루게 되면 몇 가지 문제가 발생합니다.
- 이것만으로 계산 자원을 상당히 사용하게 됩니다.
- 이 문제는 4.1 절에서 [Embedding 계층]을 도입하는 것으로 해결됩니다.
- 은닉층 이후의 계산은 [네거티브 샘플링]이라는 새로운 손실 함수를 도입해 해결합니다.

4.1.1 Embedding 계층

- 그러나 [그림 4-3]에서 결과적으로 수행하는 일은 단지 행렬의 특정 행을 추출하는 것입니다.
- 그러면 가중치 매개변수로부터 '단어 ID에 해당하는 행(벡터)'을 추출하는 계층을 만들어 봅시다. 그 계층을 Embedding 계층이라고 부르겠습니다.
- 참고로, Embedding이란 단어 임베딩이라는 용어에서 유래했습니다. 즉, Embedding 계층에 단어 임베딩(분산 표현)을 저장하는 것입니다.
- Embedding은 쉽게 말해서 행렬에서 한 행을 추출하는 과정을 의미한다.
- 이제 word2vec(CBOW 모델)의 구현은 입력 측 MatMul 계층을 Embedding 계층으로 전환할 수 있습니다!

4.2 word2vec 개선

- 은닉층의 계산을 해결은 [네거티브 샘플링]이라는 기법을 활용해 해결합니다.
- Softmax 대신 네거티브 샘플링을 이용하면 어휘가 아무리 많아져도 계산량을 낮은 수준에서 일정하게 억제할 수 있습니다.

4.2.2 다중 분류에서 이진 분류로

- 네거티브 샘플링 기법의 핵심 아이디어는 [이진 분류]에 있습니다.
- 더 정확하게 말하면, [다중 분류]를 [이진 분류]로 근사하는 것이 네거티브 샘플링을 이해하는데 중요한 포인트입니다.
- 이제부터 우리가 생각해야 할 것은 [다중 분류] 문제를 [이진 분류] 방식으로 해결하는 것입니다.
- 어렇게 하면 출력층에는 뉴런을 하나만 준비하면 됩니다. 출력층의 이 뉴런이 'Say'의 점수를 출력하는 것이죠. (Embedding처럼 하나의 score을 도출하는 과정이 네거티브 샘플링이다)
- [그림 4-8]처럼 출력 측의 가중치 $W_{아웃}$에서는 각 단어 ID의 단어 벡터가 각각의 열로 저장되

어 있습니다. 이 예에서는 'say'에 해당하는 단어 벡터를 추출합니다. 그리고 그 벡터와 은닉층 뉴런과의 내적을 구합니다.
- 이전까지의 출력층에서는 모든 단어를 대상으로 계산을 수행했습니다. 하지만 여기에서는 'say'라는 단어 하나에 주목하여 그 점수만을 계산하는 게 차이입니다. 그리고 시그모이드 함수를 이용해 그 점수를 확률로 변환합니다.

4.2.3 시그모이드 함수와 교차 엔트로피 오차

- 이진 분류 문제를 신경망으로 풀려면 점수에 시그모이드 함수를 적용해 확률로 변환하고, 손실을 구할 때는 손실 함수로 교차 엔트로피 오차를 사용합니다.
- 다중 분류의 경우, 출력층에서는 소프트 맥스 함수를, 손실 함수로는 교차 엔트로피 오차를 이용합니다. 이진 분류의 경우, 출력층에서는 시그모이드 함수를, 손실 함수로는 교차 엔트로피 오차를 이용합니다.
- 그리고 그 오차가 앞 계층으로 흘러가므로, 오차가 크면 크게 학습하고, 오차가 작으면 작게 학습합니다.

4.2.4 다중 분류에서 이진 분류로 (구현)

- 앞 절에서는 Embedding 계층을 구현했습니다. 이 계층은 대상 단어 ID의 분산 표현 (단어 벡터)을 추출합니다.

4.2.5 네거티브 샘플링

- 지금까지 배운 것으로 주어진 문제를 다중 분류에서 이진 분류로 변환할 수 있습니다. 하지만 안타깝게도 이것만으로는 문제가 다 해결되지 않습니다. 지금가지는 긍정적인 예(정답)에 대해서만 학습했기 때문이죠. 다시 말해 부정적인 예를 입력하면 어떤 결과가 나올지 확실하지 않습니다.
- 그러면 모든 부정적 예를 대상으로 하여 이진 분류를 학습시켜보면 어떨까요? 대답은 물론 [아니오]입니다.
- 그래서 근사적인 해법으로, 부정적 예를 몇 개 선택합니다. 즉, 적은 수의 부정적 예를 샘플링해 사용합니다. 이것이 바로 '네거티브 샘플링' 기법이 의미하는 바입니다.
- 정리하면, 네거티브 샘플링 기법은 긍정적 예를 타깃으로 한 경우의 손실을 구합니다. 그와 동시에 부정적 예를 몇 개 샘플링(선별)하여, 그 부정적 예에 대해서도 마찬가지로 손실을 구합니다. 그리고 각각의 데이터(긍정적 예와 샘플링된 부정적 예)의 손실을 더한 값을 최종 손실로 합니다.

4.2.6 네거티브 샘플링의 샘플링 기법

- 부정적 예를 어떻게 샘플링하느냐 하는 것인데, 다행히 단순이 무작위로 샘플링하는 것보다 좋은 방법이 있답니다. 바로 말뭉치의 통계 데이터를 기초로 샘플링하는 방법이죠.
- 구체적으로 말하면, 말뭉치에서 자주 등장하는 단어를 많이 추출하고 드물게 등장하는 단어를 적게 추출하는 것입니다.
- 말뭉치에서의 단어 빈도를 기준으로 샘플링하려면, 먼저 말뭉치에서 각 단어의 출현 횟수를 구해 [확률 분포]로 나타냅니다. 그런 다음 그 확률분포대로 단어를 샘플링하면 됩니다.
- 확률분포대로 샘플링하므로 말뭉치에서 자주 등장하는 단어는 선택될 가능성이 높습니다. 같은 이유로, '희소한 단어'는 선택되기 어렵습니다.

- 그런데 우연히도 '희소한 단어'만 선택되었다면 어떻게 될까요? 당연히 결과도 나빠질 것입니다. 즉, 드문 단어를 잘 처리하는 일은 중요도가 낮습니다. 그보다는 흔한 단어를 잘 처리하는 편이 좋은 결과로 이어질 것입니다.
- 이처럼 낮은 확률의 단어가 (조금 더) 쉽게 샘플링되도록 하기 위한 구제 조치로써 '0.75 제곱'을 수행합니다. 참고로 0.75라는 수치에는 이론적인 의미는 없으니 다른 값으로 설정해도 됩니다.
- 학습이 끝나면 가중치를 꺼내, 나중에 이용할 수 있도록 파일에 보관합니다. 파일로 저장할 때는 파이썬의 '피클' 기능을 이용합니다. 피클은 파이썬 코드의 객체를 파일로 저장하는 데 이용할 수 있습니다.
- word2vec으로 얻은 단어의 분산 표현은 비슷한 단어를 가까이 모을 뿐 아니라, 더 복잡한 패턴을 파악하는 것으로 알려져 있습니다. 대표적인 예가 'king - man + woman = queen'으로 유명한 유추 문제입니다. 더 정확하게 말하면, word2vec의 단어 부산 표현을 사용하면 유추 문제를 벡터의 덧셈과 뺄셈으로 풀 수 있다는 뜻입니다. 실제로 유추 문제를 풀려면 [그림 4-20]처럼 단어 벡터 공간에서 'man -> woman'벡터와 'king -> ?' 벡터가 가능한 한 가까워지는 단어를 찾습니다.
- 이처럼 word2vec으로 얻은 단어의 분산 표현을 사용하면, 벡터의 덧셈과 뺄셈으로 유추 문제를 풀 수 있습니다. 단어의 단순한 의미뿐 아니라 문법적인 패턴도 파악할 수 있는 것이죠.

4.4 word2vec 남은 주제

4.4.1 word2vec을 사용한 애플리케이션의 예

- [전이 학습] : 전이 학습은 한 분야에서 배운 지식을 다른 분야에도 적용하는 기법입니다.
- 자연어 문제를 풀 때 word2vec의 단어 분산 표현을 처음부터 학습하는 일은 거의 없습니다. 그 대신 먼저 큰 말뭉치로 학습을 끝난 후, 그 분산 표현을 각자의 작업에 이용하는 것이죠. 예컨대 텍스트 분류, 문서 클러스터링, 품사 태그 달기, 감정 분석 등 자연어 처리 작업이라면 가장 먼저 단어를 벡터로 변환하는 작업을 해야 하는데, 이때 학습을 미리 끝낸 단어의 분산 표현을 이용할 수 있습니다. 그리고 이 학습된 분산 표현이 방금 언급한 자연어 처리 작업 대부분에 훌륭한 결과를 가져다줍니다!
- 문장을 고정 길이 벡터로 변환하는 방법은 활발하게 연구되고 있는데, 가장 간단한 방법은 문장의 각 단어를 분산 표현으로 변환하고 그 합을 구하는 것입니다. 이를 bag-of-words라 하여, 단어의 순서를 고려하지 않는 모델입니다. 또한 5장에서 설명하는 순환 신경망(RNN)을 사용하면 한층 세련된 방법으로 문장을 고정 길이 벡터로 변환할 수 있습니다.
- 단어나 문장을 고정 길이 벡터로 변환할 수 있다는 점은 매우 중요합니다. 자연어를 벡터로 변환할 수 있다면 일반적인 머신러닝 기법(신경망이나 SVM 등)을 적용할 수 있기 때문입니다.
- [그림 4-21]의 파이프라인에서는 단어의 분산 표현 학습과 머신러닝 학습은 서로 다른 데이터셋을 사용해 개별적으로 수행하는 것이 일반적입니다.
- 다만, 직면한 문제의 학습 데이터가 아주 많다면 단어의 분산 표현과 머신러닝 시스템 학습 모두를 처음부터 수행하는 방안도 고려해볼 수 있습니다.
- ex. 메일 분류 데이터 분석 과정

  : 데이터(메일)를 수집 - 모은 메일들에 수동으로 레이블을 붙임 - 레이블링 작업이 끝나면 학습된 word2vec을 이용해 메일을 벡터로 변환합니다 - 그런 다음 감정 분석을 수행하는 어떤 분류 시스템(SVM이나 신경망 등)에 벡터화된 메일과 감정 레이블을 입력하여 학습을 수행합니다.

4.4.2 단어 벡터 평가 방법

- word2vec을 통해 단어의 분산 표현을 얻을 수 있었습니다. 그러면 그 분산 표현이 좋은지는 어떻게 평가할까요? 이번 절에서는 단어의 분산 표현을 평가하는 방법을 간략하게 설명합니다.
- 여러 시스템이란, 단어의 분산 표현을 만드는 시스템(word2vec)과 특정 문제에 대해 분류를 수행하는 시스템(예컨대 감정을 분류하는 SVM 등)입니다.
- 그래서 단어의 분산 표현의 우수성을 실제 애플리케이션과는 분리해 평가하는 것이 일반적입니다.
- 자주 사용되는 평가 척도가 단어의 '유사성'이나 '유추 문제'를 활용한 평가입니다.
- 단어의 유사성 평가에서는 사람이 작성한 단어 유사도를 검증 세트를 사용해 평가하는 것이 일반적입니다.
- 사람이 단어 사이의 유사한 정도를 규정합니다. 그리고 사람이 부여한 점수와 word2vec에 의한 코사인 유사도 점수를 비교해 그 상관성을 보는 것입니다.
- 1) 모델에 따라 정확도가 다릅니다 (말뭉치에 따라 적합한 모델을 선택)
  2) 일반적으로 말뭉치가 클수록 결과가 좋습니다 (항상 데이터가 많은 게 좋음)
  3) 단어 벡터 차원 수는 적당한 크기가 좋습니다 (너무 커도 정확도가 나빠짐)
- 그러므로 유추 문제를 정확하게 풀 수 있는 단어의 분산 표현이라면 자연어를 다루는 애플리케이션에서도 좋은 결과를 기대할 수 있을 것입니다. (왜냐, 학습 데이터와 실제 데이터는 다르기에)
- 다만, 단어의 분산 표현의 우수함이 애플리케이션에 얼마나 기여하는지는 애플리케이션 종류나 말뭉치의 내용 등, 다루는 문제 상황에 따라 다릅니다. 즉, 유추 문제에 의한 평가가 높다고 해서 여러분의 애플리케이션에서도 반드시 좋은 결과가 나오리라는 보장은 없습니다.

4.5 정리

- 이번 장에서는 word2vec 고속화를 주제로 앞 장의 CBOW 모델을 개선했습니다.
- 구체적으로 Embedding 계층을 구현하고 네거티브 샘플링이라는 새로운 기법을 도입했습니다.
- 이렇게 한 배경에는 말뭉치의 어휘 수 증가에 비례해 계산량이 증가하는 문제가 있었습니다.
- 이번 장에서의 핵심은 '모두' 대신 '일부'를 처리하는 것입니다.
- 네거티브 샘플링은 '모든' 단어가 아닌 '일부' 단어만을 대상으로 하는 것으로, 계산을 효율적으로 수행해줍니다.
- word2vec의 사상은 자연어뿐 아니라 다른 분야에도 응용되고 있습니다.
- 네거티브 샘플링은 부정적 예를 몇 개 샘플링하는 기법으로, 이를 이용하면 다중 분류를 이진 분류처럼 취급할 수 있다.