텍스트를 다루는 방법에 대해서 발표하겠습니다.

텍스트 원본을 딥러닝 모델의 입력으로 사용하지 못하기 떄문에 수치형 텐서로 바꿔줘야 합니다. 이때 텍스트를 수치형 텐서로 변환하는 과정을 텍스트 벡터화라고 합니다.

텍스트 벡터화에는 여러가지 방식이 있습니다. 첫번쨰는 텍스트를 단어로 나누는 법, 그 다음은 텍스트를 문자로 나누는 법, 그리고 마지막으로 앤 그램을 추출하는 방법이 있습니다. 텍스트를 나누는 단위를 토큰이라고 하는데요 옆에서 보시면 더 캣 온 더 맷이라는 문장을 단어 단위로 나눠놨습니다. 이 과정을 토큰화라고 하며 각 단어는 토큰이라고 불립니다. 그리고 이것을 수치형 벡터로 연결하는 것을 벡터화라고 하는데 이 과정에서 원핫인코딩이나 단어임베딩이 쓰입니다.

일단 단어 수준의 원핫인코딩을 보겠습니다. 원핫인코딩이란 특정 단어를 제외하고 나머지를 0으로 채우는 과정을 말합니다. 코드를 보겠습니다. 여기 샘플 문장이 두 개가 있습니다. 스플릿을 사용해 단어 단위로 쪼갭니다. 그 이후에 단어마다 고유한 인덱스를 토큰인덱스에 할당합니다. 여기서 인덱스 0은 사용하지 않기때문에 뒤에 +1이 붙습니다. 그 다음 제로 메소드를 사용해 나머지를 0으로 채우는 과정을 거칩니다.

결과를 보시면 the cat sat on the mat은 차례대로 원핫인코딩이 되어있는 것을 볼 수 있습니다. 그리고 아래 the dog ate my homework를 보시면 첫번째 문장과 the 단어는 똑같으므로 원핫인코딩이 첫번째 인덱스에 마킹이 되어있는것을 볼 수 있습니다. 그 이후엔 mat 이후의 인덱스에 차례대로 마킹이 되어있습니다. 첫번째 문장에서 the가 두번이 들어갔는데 다른 위치에 원핫인코딩 되어있는 이유는 대문자 때문입니다. 이러한걸 방지하기 위해 보통은 대문자를 다 소문자로 바꾸고 벡터화를 시킵니다.

다음은 문자 수준의 원핫인코딩입니다. 여기서 characters를 출력해보면 아래와 같은 아스키 문자가 출력이 됩니다. 이것을 사용해 원핫인코딩을 하는건데요

결과를 보겠습니다. 지금 프린트 하는건 the cat sat on the mat에서 맨 앞 문자인 대문자 t를 출력한 것입니다. 보시면 56번 인덱스에 마킹이 되어있는것을 볼 수 있습니다.

케라스에는 문자 수준의 원핫인코딩으로 변환해 주는 유틸리티가 있습니다. 여기서 보시면 tokenizer 객체를 만든 후, fit on texts를 사용해 단어 인덱스를 구축합니다. 그 이후 원핫인코딩을 합니다. 결과를 보시면 9개의 각자 다른 단어를 발견했다고 합니다. The는 대문자 소문자 관계없이 같은 단어 인덱스로 판단하는것을 볼 수 있습니다.

다음은 해싱 기법을 사용한 원핫인코딩도 있는데요 여기선 인덱스를 딕셔너리에 저장하는 대신 단어를 해싱하여 고정된 크기의 벡터로 변환합니다. 메모리를 절약할 수 있는 장점이 있지만 해시 충돌이 일어날 수 있습니다. 여기선 1에서 1000까지 랜덤한 정수 인덱스로 반환합니다. 크기가 1000인 벡터로 저장해 벡터가 길어 아래 결과는 잘랐지만 1로 마킹이 되어있는 것을 볼 수 있습니다.

다음은 단어 임베딩을 사용할건데요. 원핫인코딩은 보기 편하고 희소성을 띄고있지만 고차원이기 때문에 메모리를 많이 써야한다는 단점이 있습니다. 이것을 보완한 방법이 바로 단어 임베딩입니다. 밀집 벡터이고 저차원이라 메모리를 절약할 수 있습니다. 원핫인코딩이랑은 달리 단어 임베딩은 데이터로부터 학습합니다.

단어 임베딩을 만드는 방법은 두가지인데 한개는 단어 임베딩을 처음부터 학습하는 것이고, 두번째는 사전 훈련된 단어 임베딩을 로드 하는것입니다.

단어 벡터 사이에 관계를 얻으려면 단어 사이에 있는 관계를 파악해야 합니다. 즉, 비슷한 단어는 거리를 가까이 임베딩해 임베딩 공간을 구축하는 것입니다. 여기서 보시면 cat에서 tiger, dog에서 wolf로 이동하는 것을 같은 벡터로 나타낼 수 있습니다. 이것은 애완동물에서 야생동물로 이동하는 것으로 해석할 수 있습니다. 이번엔 wolf에서 tiger로 이동하는 것을 벡터로 나타내면 개과에서 고양이과로 이동하는 벡터로 해석할 수 있습니다. 이런식으로 기하학적 변환을 하는데 보통은 복수벡터와 성별벡터를 많이 씁니다.

이제 코드를 보겠습니다. 임베딩 층을 만들고 데이터를 로드합니다.

여기서 모델을 만들었습니다. Flatten 층을 사용해 3d 텐서를 2d 텐서로 펼칩니다. 즉 평평하게 만든다고 할 수 있겠죠. 그 이후에 학습을 시켜보면

75%의 정확도가 나옵니다. 하지만 이 모델은 단어 사이의 관계나 문장 구조를 고려하지 않았습니다. 그래서 임베딩 층 위에 순환 층이나 1d 합성곱층을 추가하는 것이 좋습니다.

여기에선 사전 훈련된 단어 임베딩을 사용하겠습니다. Imdb 데이터는 미리 토큰화가 되어있기 떄문에 원본 데이터셋을 이 링크에서 내려받고 압축을 해제합니다.

그 이후에 데이터를 토큰화 시킵니다. 사전 훈련 된 단어 임베딩이기 때문에 훈련 데이터는 200개만 잡겠습니다.

그 이후에 이 링크에서 사전 계산 된 임베딩을 받습니다. 이 파일을 압축해제하면 txt파일이 나오는데 이 파일을 파싱하여 단어와 이에 상응하는 벡터 표현을 매핑하는 인덱스를 만드는 과정입니다.

여기선 임베딩 층에 넣을 수 있도록 임베딩 행렬을 만드는 과정입니다.

모델을 정의하고 학습을 돌리겠습니다. 여기선 dense층이 두개가 됐습니다

모델 층에 glove 임베딩을 로드하는 과정입니다. 가중치값을 넣어서 glove 행렬을 로드합니다.

그 다음 훈련을 시켜보고 그래프를 그려보면 과대적합이 나타나는 것을 볼 수 있습니다. 훈련 샘플 수가 적기때문에 일어나는 현상입니다. 사전 훈련된 단어 임베딩을 사용하지 않거나 임베딩 층을 동결하지 않으면 같은 모델을 훈련할 경우 해당 작업에 특화된 모델이 만들어 질 것입니다. 사전 훈련된 모델을 사용하지 않으면

비슷한 정확도를 가집니다. 책에서는 사전 훈련된 단어 임베딩보다 정확도가 낮게 나왔다고 하니 여기서는 사전 훈련된 단어 임베딩을 사용하는 것이 좋을것 같습니다.

그래프를 보니 확실히 앞 그래프보다 처참한 결과를 보이고 있습니다.

이번엔 훈련 샘플 수를 2000개로 늘려서 확인해보겠습니다.

정확도가 70퍼가 된것을 볼 수 있습니다. 그래프도 아까보다는 많이 좋아졌습니다.

이제 마지막으로 테스트 데이터에서 모델을 평가해보겠습니다. 일단 테스트 데이터를 토큰화 시킵니다. 그 다음 첫번째 모델을 로드해 평가해보니 테스트 정확도는 50퍼 정도입니다.

감사합니다.