

Word Embedding Vector

Word Embedding

이전 챕터에서 단어의 의미와 모호성에 대해 다루었습니다. 이전에 다루었듯이 사람의 언어는 discrete한 형태의 단어로 이루어져 있습니다. 내부의 의미는 단어와 단어끼리 연관성이 있을 수 있지만, 형태가 다른 경우에는 곁에서 보기에는 연관성이 얼마나 있는지 (특히 컴퓨터의 경우에는) 알기 쉽지 않습니다. 이런 자연어의 특성 때문에, 자연어처리에서 단어 또는 문장(문서)를 벡터(vector)로 나타내는 것은 매우 큰 숙명이었습니다. 비록 자연어의 형태와 속성이 사람이 처리하기는 쉬운 형태로 발전해왔지만, 컴퓨터가 이해하고 처리하기에는 어려운 형태이기 때문입니다. 즉, 사람이 사용하는 자연어의 형태와 컴퓨터가 이해하는 벡터로 변환이 가능한 함수 또는 맵핑 테이블(mapping table)을 만들어 내는 과정은 매우 중요합니다.

따라서 우리는 이전 챕터에서 단어의 의미에 대해 다루면서, 코퍼스로부터 단어의 특징(feature)를 추출하여 벡터로 만드는 과정을 다루어보았습니다. 예를 들어 우리는 코퍼스에서 정해진 크기의 윈도우 내에 함께 나타난 출현 빈도를 세어 매트릭스로 나타내기도 하였습니다. 하지만 우리는 차원의 저주(curse of dimensionality)로 인해서, 그 결과물은 여전히 sparse한 벡터가 나타나는 것을 보았습니다. 같은 데이터를 표현하는데 있어서 가능한 낮은 차원으로 표현할수록 쉽게 모델링하고 학습할 수 있기 때문에, 이러한 sparse한 벡터로 나타나는 것 보단 dense한 벡터로 표현해 주는 것이 훨씬 좋을 것입니다.

이번 챕터에서는 이처럼 단어를 컴퓨터가 이해하고 처리하기 쉬운 형태로 변환하는 과정인 word embedding에 대해 다루어 보겠습니다. # One-hot encoding

단어는 discrete한 심볼(symbol)로 그 내부의 의미는 유사성이 있을 수 있지만, 곁 형태는 다른 경우가 많습니다. (동형이의어 제외) 따라서 가장 기본적으로 단어를 벡터로 나타내는 방법은 one-hot 인코딩(encoding)이라는 방식입니다. 이 방식은 말 그대로 단 한개의 1과 나머지 수많은 0들로 표현된 인코딩 방식을 뜻합니다. One-hot 인코딩 벡터의 차원은 보통 전체 어휘(vocabulary)의 갯수가 되며, 당연히 보통 그 숫자는 매우 큰 숫자가 됩니다. (대략 30,000~100,000)

$$v \in \mathbb{R}^{|V|}, \text{ where } v \text{ is one-hot vector and } |V| \text{ is vocabulary size.}$$

따라서 전체 단어에 대해서 one-hot 벡터를 구성한다면 아래와 같은 형태가 될 것입니다.



Figure 1: Tomas Mikolov – Image from web

단어	사전에서의 순서(index)	One-hot 벡터
...		
강아지	8	0, 0, 0, 0, 0, 0, 1, 0, 0, 0, ..., 0
개	9	0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ..., 0
고양이	10	0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ..., 0
구렁이	11	0, 0, 0, 0, 0, 0, 0, 0, 0, 1, ..., 0
...		
하마	20,567	0, ..., 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
...		

위처럼 사전(dictionary)내의 각 단어를 one-hot encoding 방식을 통해 벡터로 나타낼 수 있습니다. 문제는 표현하는 정보에 비해 벡터의 차원이 너무 커진 것입니다. 수만개의 단어 중 하나의 단어를 표현하기 위해서 단어수 만큼의 차원의 벡터에서 하나의 차원을 제외하고 모두 0으로 채웠기 때문입니다. 즉, 하나의 차원을 제외한 모든 차원이 0으로 채워진 벡터입니다. 이러한 벡터를 우리는 sparse vector라고 부릅니다.

Curse of dimensionality

문제는 이런 sparse vector는 기계학습에 있어서 매우 큰 장벽으로 작용한다는 점입니다. 예를 들어, 점보를 표현하는데 훨씬 큰 차원이 사용되었다면 작은 차원으로 같은 정보를 표현한 것에 비해서, 상대적으로 같은 크기의 공간에 표현되는 정보는 훨씬 더 적을 것이기 때문입니다.

차원의 저주: 차원이 높을 수록 같은 정보를 표현하는데 불리합니다.

우리는 이런 문제를 차원의 저주(curse of dimensionality)라고 부릅니다.

Similarity

One-hot encoding을 통해 sparse vector로 표현된 워드 임베딩 벡터의 경우에는 또 다른 어려움이 있습니다. 바로 one-hot encoding을 통해서는 단어간의 유사도를 표현할 수 없다는 것입니다.

$$[0, 0, \dots, 1, 0] \times [0, 1, 0, \dots, 0]^T = 0$$

우리는 임베딩 벡터간의 유사도 비교를 통해, 부족한 정보를 비슷한 다른 단어로부터 가져올 수 있을 것 입니다. 하지만 one-hot encoding을 통해 구한 임베딩 벡터의 경우에는 코사인 유사도(cosine similarity)가 항상 0일 수 밖에 없습니다. 또한, 유클리디안 거리(eucleadian distance)의 경우에도 모두가 같을 것 입니다. 따라서, one-hot encoding에 따르면, 컴퓨터는 ‘고양이’와 ‘강아지’, 그리고 ‘개’ 사이의 유사도를 구할 수 없을 것 입니다. 이것은 데이터를 통해서 학습을 수행하는 기계학습의 특성상 데이터가 많을 수록 유리한데 반해, 불리하게 작용할 수 밖에 없습니다. ‘강아지’에 대한 데이터가 적어 일반화(generalization)하기 어려울 때, ‘개’와 관련된 데이터로부터 도움을 받을 수 없을 것이기 때문입니다.

따라서 우리는 위와 같이 one-hot encoding의 한계를 실감할 수 밖에 없습니다. 따라서 우리는 차원의 저주로부터 벗어나기 위해 차원을 축소하여 단어를 표현해야 할 필요성을 느낍니다. # Dimension Reduction

이전 장에서 우린 높은 차원에서 데이터를 표현하였기 때문에 어려움이 많은 것을 살펴보았습니다. 따라서 우리는 같은 정보를 표현하기 위해서 보다 낮은 차원을 사용하는 것이 중요합니다.

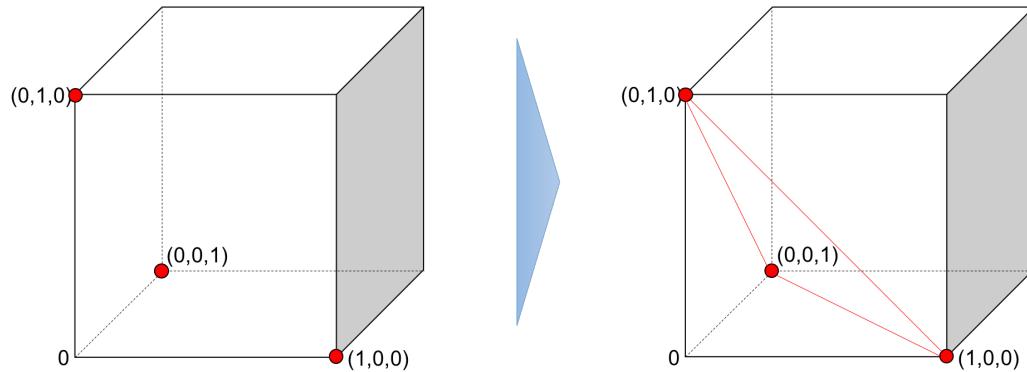


Figure 2: One-hot 벡터의 경우에는 아래와 같이 차원 축소가 가능할 것입니다.

이번 장에서는 좀 더 작은 차원으로 효율적으로 정보를 표현하는 방법에 대해서 다루어 보겠습니다.

Principal Component Analysis (PCA)

대표적인 차원축소 방법으로는 PCA(Principal Component Analysis, 주성분분석)가 있습니다.

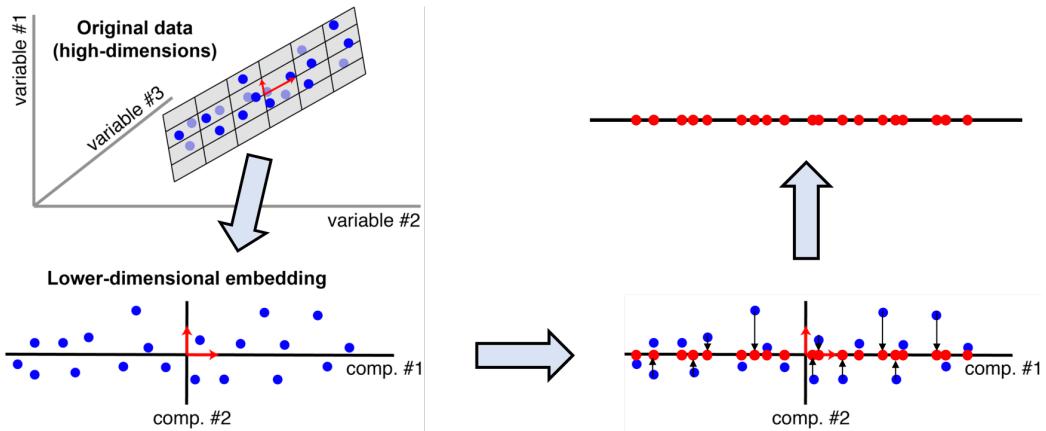


Figure 3: 3차원에서 2차원, 다시 1차원으로 PCA를 수행하는 예

위와 같이 고차원(high-dimension)의 데이터를 보다 낮은 차원으로 표현하는 것이 가능합니다. 주로 Singular Vector Decomposition (SVD)를 통해 PCA를 수행 할 수 있습니다.

이때 축소를 위한 주성분(principal component)는 위와 같은 조건을 만족합니다. 고차원에서 주어진 데이터들을 임의의 주성분 hyper-plane에 투사(projection)하였을 때, 투사점 사이가 최대한 멀어지도록 되어야 합니다. 또한, hyper-plane으로 투사할 때, 원래 벡터와 hyper-plane상의 거리가 최소가 되도록 하여야 합니다.

PCA를 통해 우리는 효과적으로 고차원의 데이터를 보다 낮은 차원으로 압축할 수 있습니다. 하지만 위에서 언급했듯이, 실제 데이터(점)의 위치와 hyper-plane에 투사된 점의 거리가 생길 수 밖에 없습니다. 이는 곧 정보의 손실을 의미합니다. 특히 주성분은 직선 또는 평면일 수 밖에 없기 때문에, 이러한 손실을 최소화 하는 것은 생각보다 어렵습니다. 너무 많은 정보가 손실된다면, 효율적으로 정보를 학습하거나 복구할 수 없기 때문입니다. 따라서 높은 차원에 표현되어 있는 정보를 지나치게 낮은 차원으로 축소하여 표현하는 것은 어려움이 따릅니다. 특히 데이터가 비선형적으로 표현되어 있을수록 이는 점점 더 어려워집니다.

Manifold Hypothesis

이때 하나의 가설을 통해 우리는 좀 더 차원축소를 효율적으로 접근해 볼 수 있습니다. 사실 one-hot encoding과 같이 대부분의 높은 차원에 존재하는 데이터들은 비록 높은 차원에서 표현되어 있지만, 해당 데이터들을 아우르는 낮은

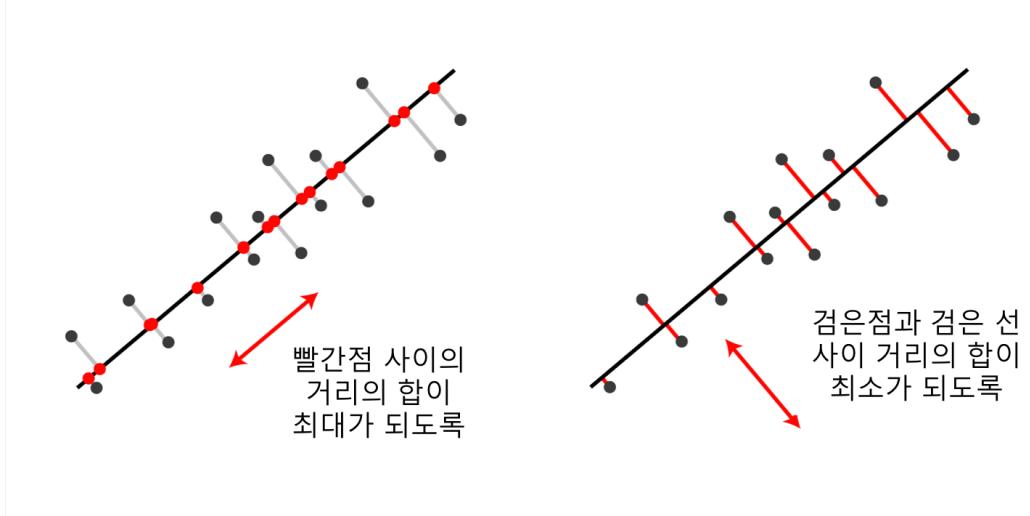


Figure 4: 주성분의 조건

차원의 매니폴드(manifold, 다양체)가 존재한다는 가설입니다.

위와 같이 3차원상에 분포한 데이터를 아우르는 소용돌이 모양의 구부려진 2차원 매니폴드가 존재할 수도 있을 것 입니다. 그럼 우리는 이런 매니폴드를 찾아 2차원의 평면 상에 데이터 포인트들을 표현할 수 있을 것 입니다. 만약 이 매니폴드를 찾을 수 있다면 이전 PCA처럼 데이터를 hyper-plane에 투사(projection)하며 생긴 손실을 더욱 최소화 할 수 있을 것 입니다.

또한, 매니폴드 가설에 따르면 또 하나의 흥미로운 특성이 있습니다. 비록 고차원상에서는 가까운 거리에 있던 벡터들일지라도, 매니폴드를 저차원 hyper-plane으로 표현하였을 때는 오히려 거리가 멀어질 수 있다는 것 입니다. 그리고 매니폴드 표면(surface)상에서 가까운 점들끼리는 실제 의미적으로 가까운 데이터를 표현하고 있다는 것 입니다.

Why deep learning is so working well

바로 딥러닝이 훌륭한 성능을 내는 이유가 여기 있습니다. 거의 모든 문제에 있어서 딥러닝이 문제를 풀기위해 수행하는 것은 바로 차원 축소이며, 그 과정은 데이터가 존재하는 고차원(high-dimension)상에서 매니폴드(manifold)를 찾는 과정 그 자체이기 때문입니다. 다른 선형적인 방식에 비해서 딥러닝은 비선형적인 방식으로 차원축소를 수행하며, 그 과정에서 해당 문제를 가장 잘 해결하기 위한 매니폴드를

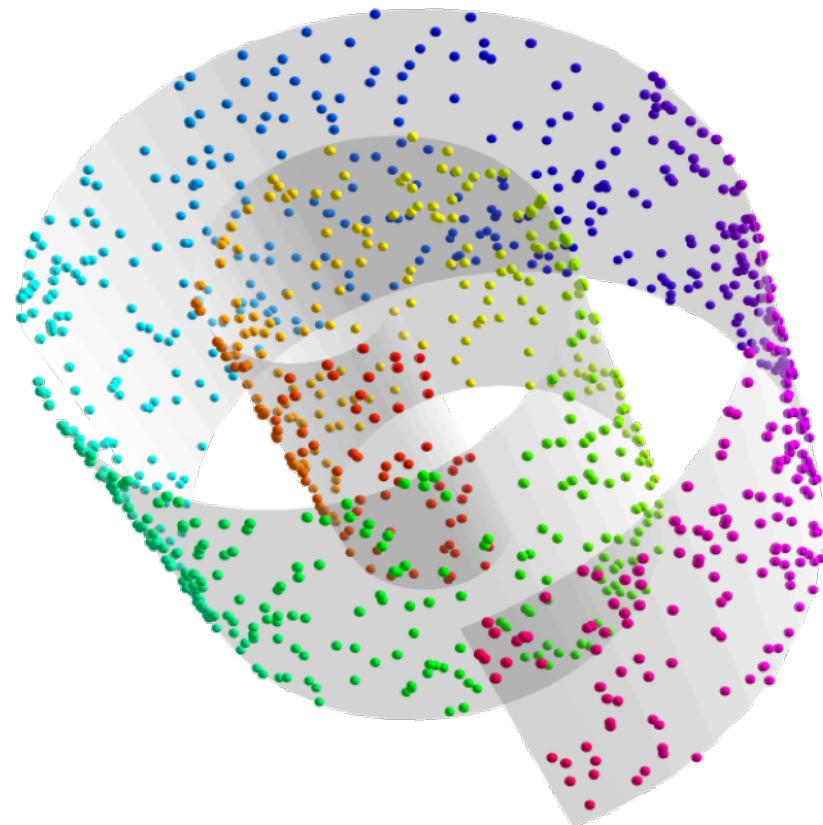


Figure 5:

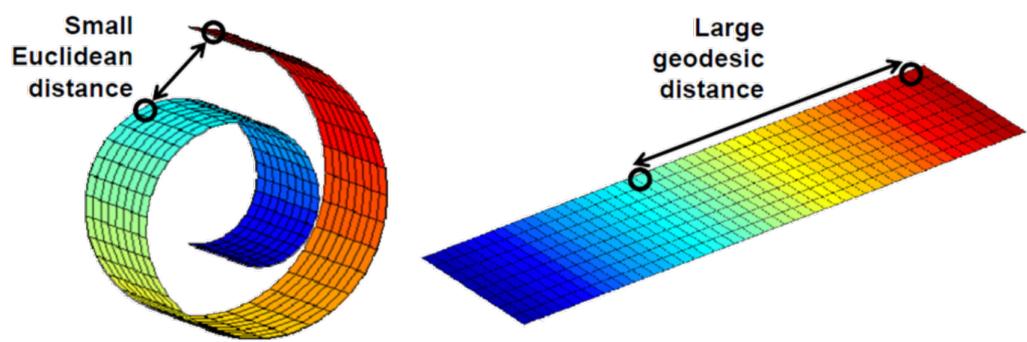


Figure 6:

자연스럽게 찾아냅니다. 이것이 바로 딥러닝이 이토록 성공적으로 동작하는 이유입니다.

Auto-encoder

그럼 본격적으로 자연어처리에서 단어를 표현하기 위한 차원 축소를 다루기에 앞서, 오토인코더(auto-encoder)에 대해 다루고 넘어가도록 하겠습니다. 오토인코더는 아래와 같은 구조를 가진 딥러닝 모델입니다.

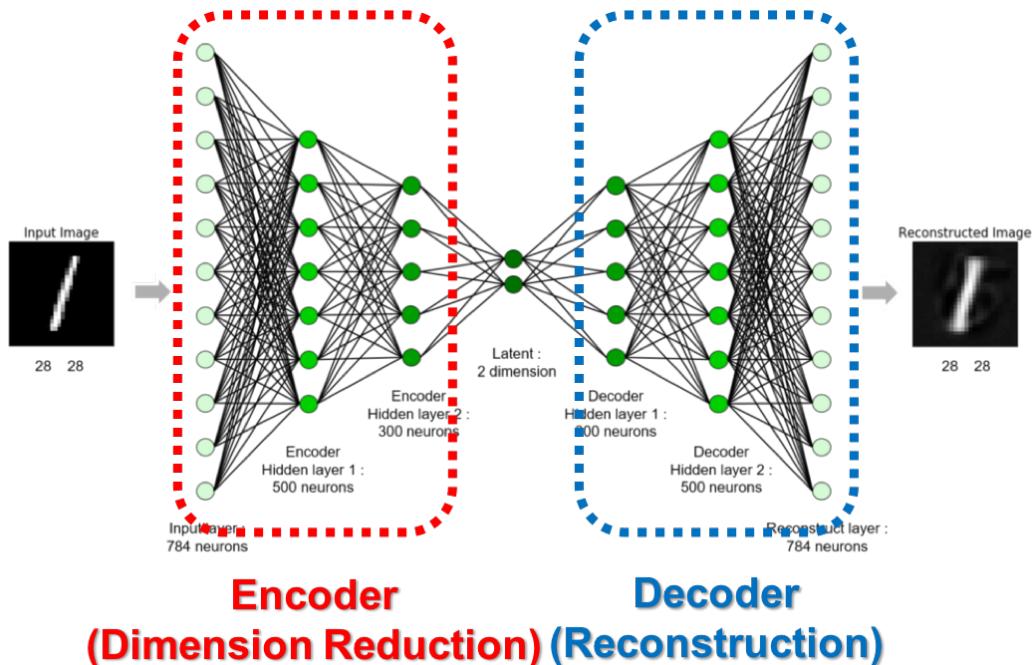


Figure 7:

고차원의 입력을 받아 저차원으로 축소하는 인코더를 거쳐 매니폴드인 bottle-neck(병목)에서의 숨겨진(latent, hidden) 데이터로 표현 합니다. 그리고 디코더는 숨겨진 저차원에서의 벡터를 받아, 다시 원래 입력 데이터가 존재하던 고차원으로 데이터를 복원하는 작업을 수행 합니다.

따라서 이 구조의 모델을 훈련할 때엔, 복원된 데이터와 실제 입력 데이터 사이의 차이를 최소화하도록 손실함수(loss function)을 구성합니다.

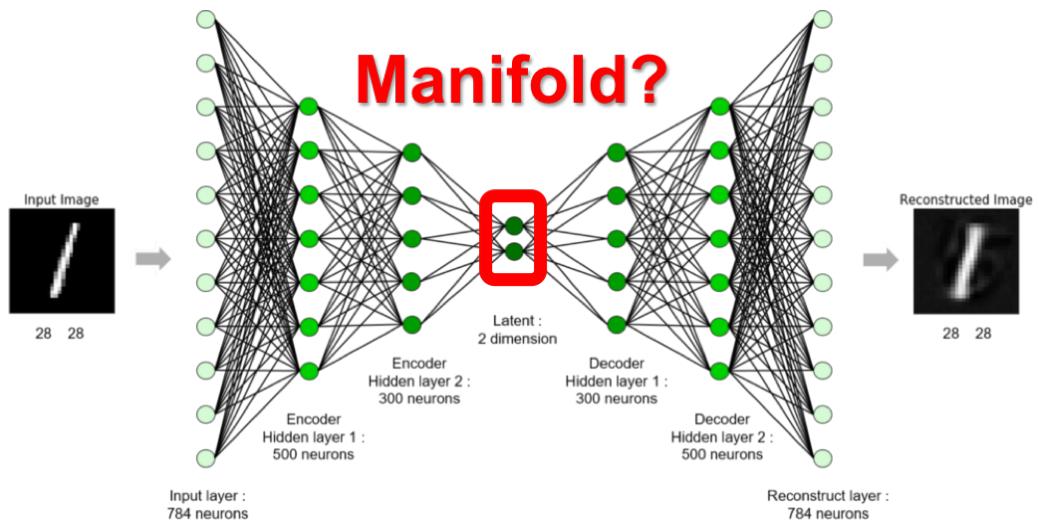


Figure 8:

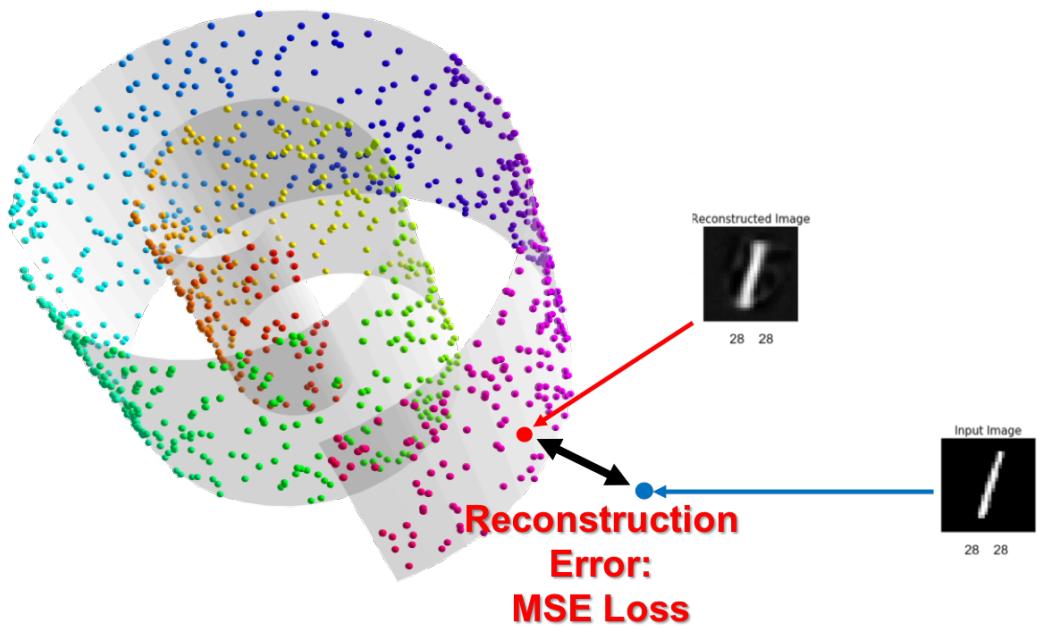


Figure 9:

하지만 고차원에서 저차원으로 데이터를 표현하는 것에 손실이 따를 수 있기 때문에, 훈련이 완료된 모델일지라도 복원된 데이터는 실제 입력과 차이가 있을 수 있습니다.

우리는 이 오토인코더를 사용하여 이전 챕터에서 구해진 sparse word feature 벡터를 dense word embedding 벡터로 변환할 수 있을 것 입니다. # Myth

우리는 이번 챕터를 통해 단어를 벡터로 표현하는 방법에 대해서 살펴보고 있습니다. 이어지는 섹션에서 skip-gram 또는 GloVe를 사용하여 One-hot encoding의 sparse 벡터를 차원 축소(dimension reduction)하여 훨씬 작은 차원의 dense 벡터로 표현하는 방법에 대해 다룰 것 입니다.

이에 앞서, 하지만 많은 분들이 헷갈려 하는 부분이 있다면, 이렇게 훈련한 단어 임베딩 벡터를 추후 우리가 다룰 텍스트 분류, 언어모델, 번역 등의 딥러닝 모델들의 입력으로 사용할 것이라고 생각한다는 점입니다.

Word2Vec을 통해 얻은 단어 임베딩 벡터는 훌륭하게 단어의 특성을 잘 반영하고 있지만, 텍스트 분류, 언어모델, 번역의 문제 해결을 위한 최적의 벡터 임베딩이라고는 볼 수 없습니다.

예를 들어 긍정/부정 분류를 하는 텍스트 분류 문제의 경우에는 ‘행복’이라는 단어가 매우 중요한 특징(feature)가 될 것이고, 이를 표현하기 위한 임베딩 벡터가 존재할 것입니다. 하지만 영화 시놉시스의 장르를 분류하기 위한 문제에서는 ‘행복’이라는 단어는 다른 특징으로 작용 될 것이며, 이 분류 문제를 위한 ‘행복’ 단어의 임베딩 벡터의 값은 이전 긍정/부정 분류 문제의 값과 당연히 달라지게 될 것입니다. 따라서 문제의 특성을 고려하지 않은 단어 임베딩 벡터는 그다지 좋은 방법이 될 수 없습니다.

How to get word embedding instead of using Word2Vec

우리는 Word2Vec을 사용하지 않더라도, 문제의 특성에 맞는 단어 임베딩을 구할 수 있습니다. PyTorch를 비롯한 여러 딥러닝 프레임워크는 Embedding Layer라는 레이어 아키텍처를 제공합니다.

이 레이어는 아래와 같이 bias가 없는 Linear Layer와 같은 형태를 갖고 있습니다.

$$y = \text{emb}(x) = Wx,$$

where $W \in \mathbb{R}^{d \times |V|}$ and $|V|$ is size of vocabulary.

쉽게 생각하면 W 는 $d \times |V|$ 크기의 2차원의 매트릭스입니다. 따라서 입력으로 one-hot vector가 주어지게 되면, W 의 특정 column(구현 방법에 따라 또는 row)만

반환하게 됩니다.

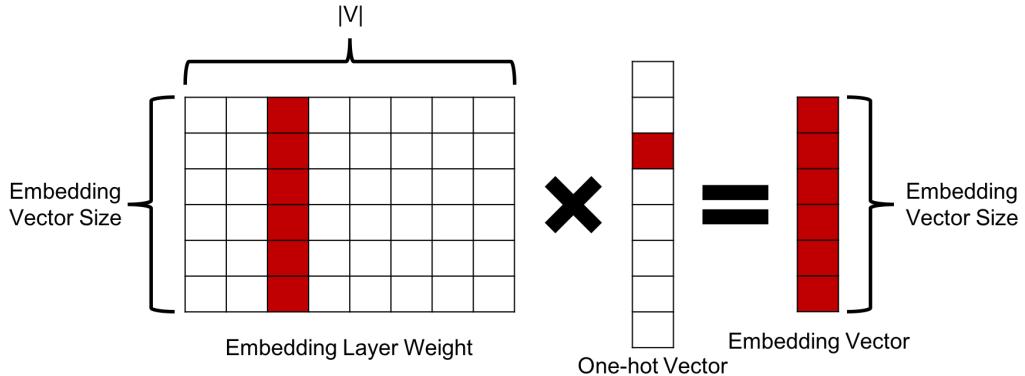


Figure 10:

따라서 최종적으로 모델으로부터 구한 손실값(loss)에 따라 back-propagation 및 gradient descent를 수행하게 되면, 자동적으로 embedding layer의 weight W 의 값을 구할 수 있게 될 것 입니다.

물론 실제 구현에 있어서는 이렇게 큰 embedding layer weight와 one-hot 벡터를 곱하는 것은 매우 비효율적이므로, 단순히 테이블에서 lookup하는 작업을 수행 합니다. 따라서 우리는 단어를 나타냄에 있어 (embedding layer의 입력으로) one-hot 벡터를 굳이 넘겨줄 필요 없이, 1이 존재하는 단어의 index 정수 값만 입력으로 넘겨주면 embedding vector를 얻을 수 있습니다.

추후 실제 앞으로 우리가 다룰 텍스트 분류나 기계번역 챕터에서 구현된 것을 살펴보면, 위에서 말한대로 embedding layer를 사용하여 구현한 것을 알 수 있습니다. # GloVe

이전 섹션에서는 Word2Vec에 대해서 다루어 보았습니다. 이번 섹션에서는 또 하나의 대표적인 word embedding 방법인 GloVe (Global Vectors for Word Representation) [Pennington et al.2014]에 대해 다루어보고자 합니다.

Arhitecture

이전에 다루었던 skip-gram은 대상 단어를 통해 주변 단어를 예측하도록 네트워크를 구성하여 word embedding 벡터를 학습하였습니다. GloVe는 대신에 대상 단어에 대해서 코퍼스에 함께 나타난 각 단어별 출현 빈도를 예측하도록 합니다. GloVe를 구하기 위한 파라미터를 구하는 수식은 아래와 같습니다.

$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{x \in \mathcal{X}} f(x) \times \|W'Wx - \log C_x\|_2$$

where C_x is vector of co-occurrences with x

Also, $x \in \{0, 1\}^{|V|}$, $W \in \mathbb{R}^{d \times |V|}$ and $W' \in \mathbb{R}^{|V| \times d}$.

Skip-gram을 위한 네트워크와 거의 유사한 형태임을 알 수 있습니다. 다만, 여기서는 classification이 아닌, 출현 빈도에 근사(approximation)하는 regression에 가깝기 때문에, Mean Square Error (MSE)를 사용한 것을 볼 수 있습니다.

마찬가지로 one-hot 인코딩 벡터 x 를 입력으로 받아 한 개의 hidden layer W 를 거쳐 output layer W' 를 통해 출력 벡터를 반환 합니다. 이 출력 벡터는 단어 x 와 함께 코퍼스에 출현했던 모든 단어들의 각 동시 출현 빈도들을 나타낸 벡터인 C_x 를 근사해야 합니다. 따라서 이 둘의 차이를 손실(loss)로 삼아 back-propagation 및 gradient descent를 통해 학습을 할 수 있습니다.

이때 단어 x 자체의 출현빈도 또는 사전확률(prior probability)에 따라서 MSE loss의 값이 매우 달라질 것 입니다. 예를 들어, C_x 가 클수록 손실(loss)값은 커질것이기 때문입니다. 따라서 $f(x)$ 는 단어의 빈도에 따라 아래와 같이 손실 함수에 가중치를 부여합니다.

$$f(x) = \begin{cases} \left(\text{Count}(x)/\text{thres}\right)^{\alpha} & \text{Count}(x) < \text{thres} \\ 1 & \text{otherwise.} \end{cases}$$

이 논문에서는 실험에 의해 $\text{thres} = 100$, $\alpha = 3/4$ 일때 가장 좋은 결과가 나온다고 언급하였습니다.

Pros

코퍼스 내에서 주변단어를 예측하고자 하는 Skip-gram과 달리, GloVe는 처음에 코퍼스를 통해 각 단어별 동시 출현 빈도를 조사하여 이에 대한 매트릭스를 만들고, 이후엔 해당 매트릭스를 통해 동시 출현 빈도를 근사(approximate)하고자 합니다. 따라서 계속하여 코퍼스 전체를 훑으며 대상 단어와 주변 단어를 가져와 학습해야 하는 skip-gram과 달리 훨씬 학습이 빠릅니다.

또한, 코퍼스를 훑으며 학습하는 skip-gram의 특성상, 사전확률(prior probability)이 작은 (즉, 출현 빈도 자체가 작은) 단어에 대해서는 학습 기회가 적을 수 밖에

없습니다. 따라서 출현 빈도가 작은 단어들은 비교적 부정확한 word embedding 벡터를 학습하게 됩니다. 하지만, GloVe는 skip-gram에 비해서 이러한 단점이 어느정도 보완되었습니다.

Conclusion

비록 GloVe의 저자는 GloVe가 가장 뛰어난 embedding 방식임을 주장하였지만, 사실 skip-gram도 파라미터 튜닝 여부에 따라 GloVe와 큰 성능 차이가 없습니다. 따라서 실제 구현하고자 할때에는 주어진 상황(예를 들어 시스템 구성)에 따라 적절한 방법을 선택하는 것도 한가지 방법입니다. # Word2Vec Practice

FastText

Visualization