TAVE 서기

|  |  |  |  |
| --- | --- | --- | --- |
| **서기 내용** | | | |
| **서기 일자** | 22.1.3 | **서기** | 8기 고성호 |
| **주제** | AutoEncoder, GAN | | |
| **시간** |  | **장소** | Zoom 미팅 |
| **스터디**  **인원** | 고성호, 권기호, 이아현, 서가을 : 시작    고성호, 권기호, 이아현, 서가을 : 종료 | | |
| **내용** | | | |
| **배운 내용** | **Chapter 17. 오토인코더와 GAN을 사용한 표현 학습과 생성적 학습**  **17.1 효율적인 데이터 표현**  **17.2 Undercomplete Linear AutoEncoder로 PCA 수행하기**  **17.3 Stacked AutoEncoder**  **17.4 Convolutional AutoEncoder**  **17.5 Recurrent AutoEncoder**  **17.6 Denoising AutoEncoder** | | |
| 17.1 효율적인 데이터 표현      **\* 위 두 개의 sequence를 완벽하게 외워야 된다고 생각해보자. 얼핏 보기엔 첫 번째 sequence가 더 짧기 때문에 더 외우기 쉬울 것으로 보인다. 하지만 자세히 들여다보면, 두 번째 sequence는 50부터 2씩 14까지 줄어드는 pattern을 가지고 있기 때문에 오히려 두 번째 sequence가 더 외우기 쉬울 것이다.**  **\* 이와 같이 긴 sequence의 경우 pattern을 찾아 기억하는 것이 더 쉽다. 우리가 사용할 autoencoder 또한 마찬가지이다.**    **\* AutoEncoder는 기본적으로 Encoder(recognition network), Decoder(generative network)부분으로 구분된다. Encoder는 입력된 정보를 latent representation으로 변환하고, Decoder는 이 latent representation을 가지고 입력 데이터를 최대한 복원한다. 이 때 encoder가 입력 데이터를 latent representation으로 변환하는 것을 앞서 숫자 sequence를 외울 때 50부터 2씩 14까지 줄어든다는 특징을 추출한 것과 연결지을 수 있다. latent representation은 입력 데이터를 더 낮은 차원의 데이터로 변환하는 과정에서 중요한 특성을 학습하고, 중요하지 않은 특성을 버리게 된다.**  **\* training 과정에서 필요한 loss의 경우 reconstruct loss라고 부르며, output과 input의 차이를 활용한다. 즉, decoder가 input을 얼마나 잘 복원했는지의 정도를 활용한다고 이해한다고 볼 수 있다.**  **\* 이렇게 latent representation이 input data보다 저차원인 구조를 띠는 autoencoder를 undercomplete autoencoder라고 부른다.**  17.2 Undercomplete Linear AutoEncoder로 PCA 수행하기  **\* AutoEncoder가 linear activation function만 활용하고, cost function이 MSE라면, 결국 PCA를 수행하는 것과 같다.**    **\* linear autoencoder를 사용해 3차원 데이터를 2차원으로 축소하기 위해 3차원 데이터를 먼저 생성하였다.**    **\* 3차원 vector를 받아 2차원 vector로 축소하는 encoder, 2차원 vector인 latent presentation을 3차원 vector로 복원하는 decoder를 구성해 20 epoch 동안 훈련했다.**      **\* predict 메소드를 사용해 encoder에 X\_train을 집어넣어 2차원 vector들을 출력하였다.**  **\* autoencoder를 통해 차원 축소를 했을 때 PCA와 동일하게 분산이 가능한 많이 보존되도록 데이터를 2차원 평면에 투영한다.**  Stacked AutoEncoder란    **\* 우리가 최초로 17.1 에서 살펴보았던 autoencoder의 구조의 경우 input layer, hidden layer, output layer로 이뤄진 가장 기본적인 autoencoder였다. 여기서 hidden layer의 개수를 늘린 것을 stacked autoencoder 혹은 deep autoencoder라고 부른다.**  **\* 이렇게 hidden layer를 늘리는 것은 autoencoder가 더 복잡한 처리를 가능하게 하겠지만, 한 편으로 일반적으로 network가 deep해질 때 생기는 overfitting등의 문제가 발생할 수 있기 때문에 과도하게 deep한 network를 구성하지 않도록 주의해야 한다.**  케라스를 이용하여 stacked autoencoder 구현하기  데이터셋 로드    **\* keras에서 제공하는 fasion mnist dataset을 활용해 stacked autoencoder를 구성해보겠다.**    **\* 구조는 앞서 살펴본 autoencoder의 hidden layer 개수를 늘린 것에 불과하기 때문에 이해하기 쉬울 것이다. 다만, 3차원 vector와는 달리, fashion mnist data는 28\*28 크기의 image가 input되었기 때문에 마지막에 28\*28로 reshape을 해준다는 것이 차이점이다.**  **\* 또한 loss로 binary crossentropy를 사용했는데, 이는 recontruction 작업을 다중 레이블 이진 분류 문제로 다루기 때문이다. 각 픽셀의 강도는 픽셀이 검정일 확률을 나타낸다.**  17.3.2 재구성 시각화      **\* input data와 output data를 비교해본 결과 원본의 형태는 알아볼 수 있지만, 디테일한 부분들은 복원되지 않은 것을 확인하였다. 이는 network를 더 deep하게 구성하거나, coding의 차원 크기를 더 크게 구성하는 등의 방법으로 개선할 수 있다.**  17.3.3 fasion MNIST 데이터셋 시각화  **\* 위 결과에서 알 수 있듯이 autoencoder를 사용한 차원 축소에는 많은 정보 손실이 발생한다. 차원 축소라는 측면에서만 보면 PCA, t-SNE 등의 차원 축소 알고리즘이 더 효율적이라고 할 수 있다.**  **\* 하지만, 다른 알고리즘들보다 autoencoder를 사용해 차원 축소를 할 때 대용량의 데이터를 처리하기 용이하다는 장점이 있다. 이러한 장단점을 활용하여 차원 축소를 진행한다면, autoencoder로 대용량의 데이터를 input으로 받아 적절한 수준까지(정보가 과다하게 손실되지 않을 수준까지) 차원 축소를 진행하고, 더욱 저차원으로 차원 축소를 진행할 때는 다른 알고리즘을 사용하는 방법을 사용할 수 있다.**      17.3.4 Stacked AutoEncoder를 이용한 비지도 사전훈련    **\* 만약 데이터의 일부만 정답 label이 존재한다면, AutoEncoder를 사용해 labeling을 시행할 수 있다.**  **\* 먼저 전체 데이터를 통해 autoencoder 전체를 훈련한다. encoder로 coding을 추출하고, coding으로 encoder의 input과 최대한 유사하게 decoder의 output을 도출하는 훈련을 통해 적절하게 차원을 축소하여 특징을 도출해내는 encoder를 구성한다. 이후 encoder 부분의 훈련된 parameter들을 복사해 새로운 모델에 적용한다.**  **\* 이 새로운 모델의 output layer에는 softmax함수를 적용한다. labeling된 데이터를 집어넣어서 softmax를 통해 올바른 class를 예측하는 훈련을 한다(만약 labeling 된 데이터의 개수가 매우 적으면, 하위 hidden layer의 가중치를 동결시키는 방법을 사용한다). 이 classifier를 가지고 labeling 되지 않은 데이터들을 labeling한다.**  17.3.5 가중치 묶기    **\* 방금 살펴본 것과 같이 autoencoder의 구조가 완벽하게 대칭이라면 encoder의 graident와 decoder의 gradient를 묶어줄 수 있다. 더 쉽게 설명하기 위해, 17장 최초에 살펴봤던 autoencoder를 예로 들어보겠다. 이 autoencoder는 3차원 vector를 집어넣어 2차원 vector로 축소하고, 2차원 vector를 다시 3차원 vector로 복원하는 모델이다. 때문에 encoder부분의 weight matrix는 3x2 matrix일 것이다. 반대로 decoder는 2차원 vector를 3차원 vector로 변환해야 하기 때문에 2x3 matrix를 weight로 사용할 것이다. 이 때 encoder가 하는 작업의 반대 작업을 decoder가 하기 때문에 단순히 encoder의 3x2 weight matrix를 전치시킨 2x3 matrix를 decoder의 weight matrix로 사용할 수 있게 된다.**  **\* 위 설명한 내용을 일반화 하자면 이렇게 표현할 수 있다. N개의 층을 가진 autoencoder의 N번째 층의 weight를 W\_L이라고 할 때 아래와 같다.**      **\* encoder부분의 weight를 transpose해서 decoder부분의 weight로 사용한 모델을 훈련시켰다.**      17.4 Convolutional AutoEncoder**1** Convolutional AutoEncoder  **\* 이미지를 다룰 경우 convolution layer를 사용해야 더 좋은 결과를 얻을 수 있다는 것을 알고 있을 것이다. 2D 이미지를 1차원 vector로 flatten해서 Dense layer에 집어넣어 연산하는 것 보다 이미지를 그대로 사용해(정확히는 그대로 사용하는 것 처럼 처리) 공간상의 정보를 보존하는 방법을 사용하는 것이 훨씬 결과가 좋다.**  **\* autoencoder도 마찬가지로 convolution layer를 사용하는 것이 이미지 처리에서 성능이 좋다. 이미지 처리를 위한 encoder decoder를 구성할 때 encoder부분에서는 일반적인 CNN 구조를 사용한다. convolution layer를 통과하고 maxpooling layer를 통과하며 이미지의 width, height는 점점 작아지고, depth는 늘어난다. 하지만 decoder 부분에서는 encoder의 작업을 반대로 해줘야 한다. 이미지의 depth를 점점 줄이고, width, height는 원본 이미지까지 늘려야 한다. 이 decoder부분에서의 연산은 keras.layers의 Conv2DTranspose로 구현 가능하다. 조금 더 이해를 돕기 위해서 이 decoder 부분의 연산에 대해 알아보자.**    **\* 일반적인 convolution layer의 연산은 위와 같다. kernel을 sparse marix로 변환한 뒤 Input vector에 곱해서 output을 도출해낸다.**      **\* 이 과정을 반대로 하기 위해서는 convolution matrix를 transpose하여 사용한다. 이를 통해 다시 원본 크기의 vector를 추출할 수 있다. 이렇게 특징 vector로부터 원본 vector를 추출해내는 것을 Transposed Convolution 이라고 한다. 우리가 사용할 decoder에서는 이 transpose convolution을 이용하게 된다. 주의할 점은 17.3에서 살펴봤던 가중치 묶기와 같이 encoder의 가중치를 decoder의 가중치와 동일하게 구성하는 것은 아니다. 반대 연산을 시행하지만, 가중치는 독립적으로 훈련된다는 점을 유의하자.**         **17.5 Recurrent AutoEncoder** **\* 만약 시계열 데이터나, 텍스트 데이터를 처리해야 할 경우라면, Dense layer로 autoencoder를 구성하는 것 보다 lstm layer를 사용하는 것이 좋은 성능을 기대할 수 있을 것이다.**    **\* timestep마다 28차원을 갖는 어떤 길이의 sequence로 처리한다.**  **\* 각 이미지를 행의 sequence로 간주하여 fashion MNIST 이미지를 처리한다.**  **17.6 Denoising AutoEncoder** Denoising AutoEncoder **\* autoencoder를 훈련시킬 때 일부러 원본에 noise를 추가하는 방식을 사용하면 더욱 효율적인 결과를 얻을 수 있게 된다. 더 자세히 설명하자면, autoencoder의 encoder부분에서 원본 데이터를 입력 받은 후 noise(Gaussian noise같은 random한 noise)를 추가하거나, dropout을 사용해 원본 데이터의 일부를 의도적으로 훼손하는 방법을 사용해 특징 vector를 구성하고, 이 특징 vector로부터 decoder가 원본 데이터를 복원하는 훈련을 하는 방법을 사용하는 것이다.**   1) Gaussian noise 사용   **\* Gaussian noise를 사용해 훈련을 진행하였다.**    **\* test시에는 정규화 층인 keras.layers.GaussianNoise()가 작동하지 않기 때문에, 따로 GaussianNoise층을 통과시킨 데이터를 test 시에 모델에 집어넣었다.**     2) Dropout 사용   **\* Gaussian noise대신 Dropout을 사용하여 훈련을 진행하였다.**    **\* Gaussian noise를 사용한 모델에서와 같이 Dropout 층도 정규화 층이기 때문에 test시에는 따로 dropout을 적용시킨 데이터를 입력해줘야 한다.** | | |
| **과제할당** | 18.1 하정현  18.2 허주희  18.3 고성호  18.4 안세윤  18.5 권기호  18.6 이문기  18.7 이아현  18.8 박제윤  18.9 서가을  18.10 권기호  18.11 고성호  18.12 이문기  18.13 안세윤  18.14 허주희 | | |
| **특이사항** | 없음 | | |
| **비고** | 없음 | | |