2017-2학기 인공지능 기말과제

Denosing Auto Encoder를 이용하여 노이즈가 섞인 손글씨(숫자) 데이터도 인식할 수 있는 숫자인식기 제작

컴퓨터과학부
2013920038
유기태

1. 주제 소개

실생활에서 숫자 인식기를 사용할 수 있는 가장 일반적인 예제로는, 은행 등에서 사용하는 전표를 들 수 있다. 적절한 전처리(Pre-Processing) 과정을 거쳐서 전표에 적힌 손글씨(숫자)를 인식시킬 수 있을 만한 데이터로 만든 다음, 인공신경망을 이용해 잘 학습된 모델에 넣어주면 해당 숫자를 자동으로 인식할 수 있을 것이다.
하지만 이러한 데이터를 취득하는 과정에서 성능이 안 좋은 구형 스캐너를 사용하거나, 데이터 전송 과정에서 노이즈가 섞일 경우, 이미지가 변질되거나 일부 손상되어 학습된 모델에서 정상적으로 인식하지 못할 수도 있다.
이러한 문제를 막기 위해, 숫자 인식기 모델 전단부에 CNN을 활용하여 잘 학습된 Denosing Auto Encoder(DAE)를 추가하여, 노이즈가 섞인 데이터도 문제없이 인식할 수 있는 숫자인식기를 구현한다.

2. 프로그램 사용 방법

- **프로그램 사용 간 주의사항**

1. 소스 파일들(*.py)과 같은 경로에 위치한 "datasets" 폴더 안에 4개의 MNIST 데이터셋 파일과 N-MNIST 데이터셋 파일이 있어야 한다.
2. module_load_data.py 파일을 모듈로 import 하여 사용하기 때문에, 파일들의 경로에 한글이 포함되면 안 된다.
3. 각 프로그램들의 실행 결과로 출력되는 이미지는 images 폴더 안에 해당 소스 파일의 이름으로 된 폴더가 생성되어, 모델의 구조나 샘플들의 처리 결과 등을 저장한다. 상위의 images 폴더는 자동으로 생성되지 않으므로 프로그램 실행 전 생성되어 있어야 한다.

- **숫자 인식기 테스트 프로그램 : Test_Digit_Recognizer.py**

테스트 샘플에 Denosing Auto Encoder를 적용하여 작동하는 숫자인식기로써, 윈도우 커맨드에서 1개의 Argument 값(0.0~1.0 사이의 실수)을 주어 실행해야 한다. 이 값은 Test Data Sample들에 노이즈를 섞을 비율이다. MNIST의 Test Data Set 에는 10,000개의 테스트 샘플이 있으므로, 예를 들어 0.3의 값을 주면, 7,000개의 정상 손글씨(숫자) 이미지와, 3,000개의 노이즈가 섞인 이미지로 테스트를 진행한다. 프로그램의 실행 결과로는, 커맨드에 에러율(Error Rate)를 출력한다.

- **MNIST 데이터 로드 모듈 : module_load_data.py**

MNIST 데이터셋을 원본 형태 그대로 불러오거나, 노이즈를 섞어서 불러오는데 사용되는 모듈

- DAE 학습 및 테스트 프로그램 : DAE_model.py

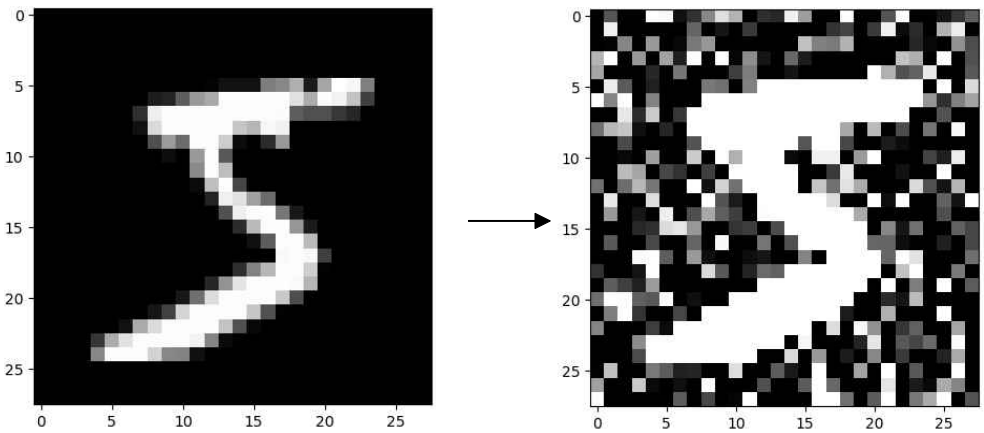
Denosing Auto Encoder 모델을 생성하고 학습하여, 학습 결과 모델 파라미터와 샘플 이미지들을 저장하는 프로그램

- 숫자 인식기 학습 프로그램 : Train_Digit_Recognizer.py

MNIST 데이터셋 중, Train Data 에 해당하는 60,000개의 정상 데이터를 불러와서, DAE 모델에 통과시킨 후, 3개의 Dense Layer를 가진 모델로 Classification이 가능하게끔 학습하는 프로그램

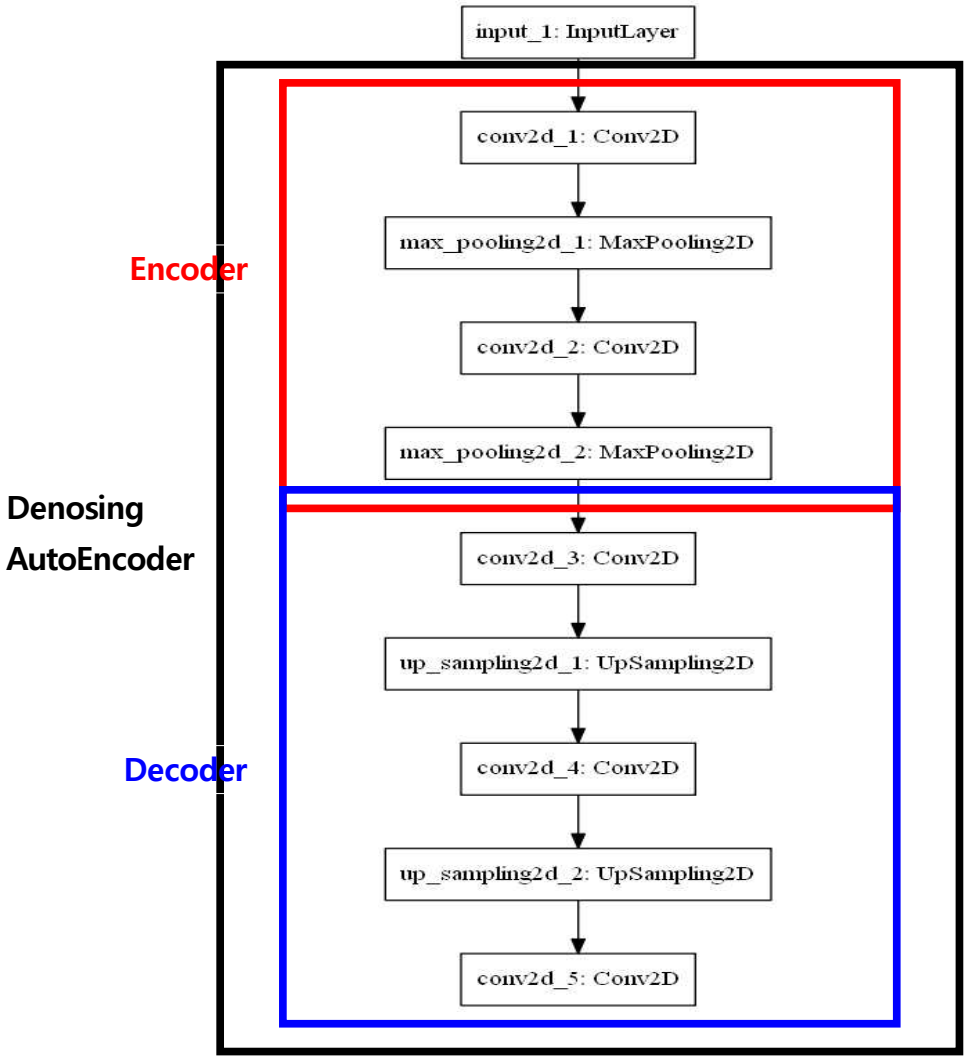
3. 구현방법

먼저, 숫자 인식기의 성능 테스트를 위해 노이즈가 포함된 데이터 샘플이 필요하다. 기존 MNIST 데이터셋에 노이즈를 추가하는 방식으로 노이즈가 추가된 샘플을 얻는데, 기존의 MNIST 데이터를 불러와 모든 데이터에 랜덤하게 Additive White Gaussian Noise를 추가한다. MNIST 데이터는 28x28x1 행렬에 0~255 사이의 숫자로 표현되어 있기 때문에, 행렬의 각 원소에 랜덤하게 랜덤 값을 더해주면 된다. 이러한 과정은 module_load_data.py 소스 파일에 구현되어 있다.



< Additive White Gaussian Noise를 섞은 MNIST 데이터 예시 >

이렇게 생성된 데이터셋을 Input으로 하여 Deep Convolutional Neural Network를 구성하여 Denosing Auto Encoder의 모델을 만든다. 노이즈를 섞기 전 데이터 샘플을 정답 레이블로 하여서 학습을 진행한다. 이 DAE의 인공신경망 구조는 아래와 같다. Neural Network 의 구현 및 학습은 Theano Backend의 Keras를 이용하였다.



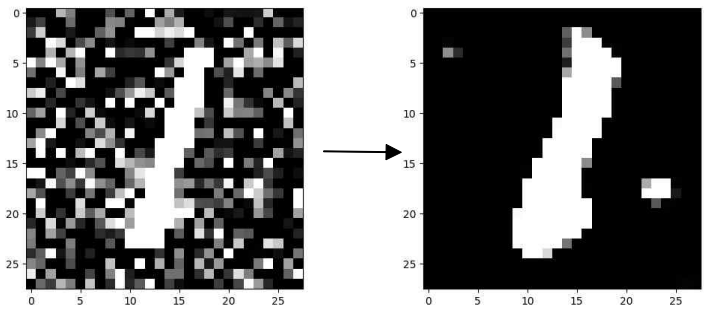
< 구현한 Denosing Auto Encoder의 인경신경망 모델 구조 >

각 2D Convolutional Layer의 필터 수는 32개로 설정했으며, 필터의 사이즈는 3x3 으로 설정했고, 모두 Zero-Padding을 하여 Layer 통과 전/후의 이미지 크기는 같다. 마지막 Output을 위한 5번째 Convolutional Layer는 Sigmoid 함수를 Activation Function으로 사용했으며, 이를 제외한 나머지 Convolutional Layer는 모두 ReLU Function을 사용했다. MaxPooling과 UpSampling의 사이즈는 2x2로 설정했다. 결과적으로, Encoder를 거친 데이터 샘플은 7x7 사이즈의 이미지 32개로 Encoding 된다.

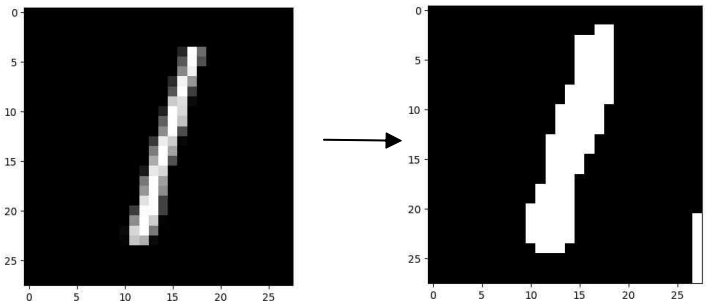
학습을 위한 Cost Function 으로는 Binary Crossentropy를 사용하였고, 파라미터 업데이트 방법으로는 Gradient Descent의 변형인 AdaDelta라는 방법을 사용하였다.
Input Data로는 Noise를 섞은 60,000개의 MNIST 데이터를 사용하였고, 정답 레이블로는 Noise를 섞기 전의 Pure한 데이터를 사용하였다. 학습은, Batch Size를 100으로 하여 Cost 감소율이 상당히 줄어드는 시점까지 하기 위해, 총 3 Epoch 동안 진행했다.

최종적으로 학습된 Denosing Auto Encoder의 성능은 다음과 같다.

< Noise가 섞인 샘플이 Encoder 부분을 거쳐 32개의 8x8 크기 이미지로 인코딩 된 모습 >

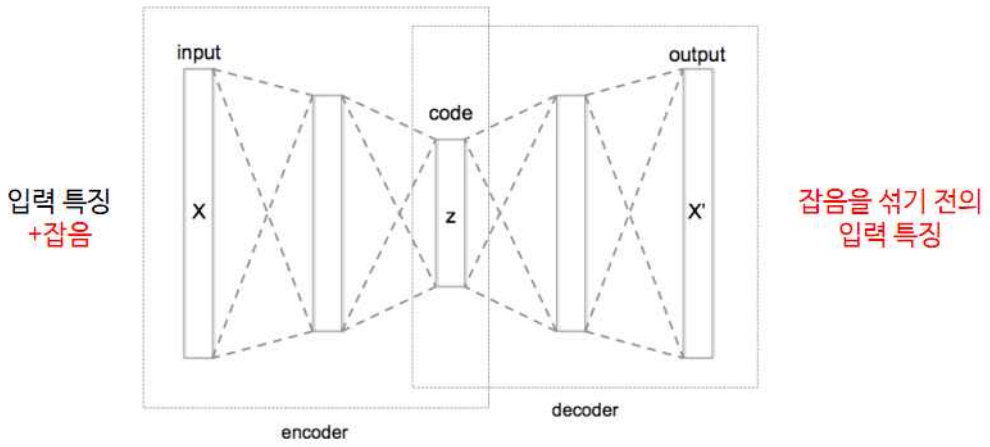


< Noise를 섞은 샘플을 Denosing Auto Encoder를 통과 시킨 최종 결과 >

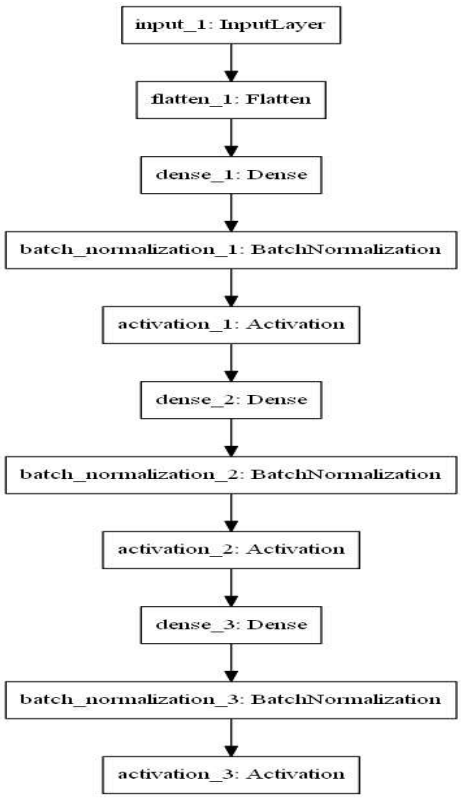


< 정상 샘플을 Denosing Auto Encoder를 통과 시킨 최종 결과 >

위 페이지의 예시들을 통해 본 강의 12주차 강의노트에서 제시된 아래의 DAE 모델이 정상적으로 구현, 학습되었음을 알 수 있다.

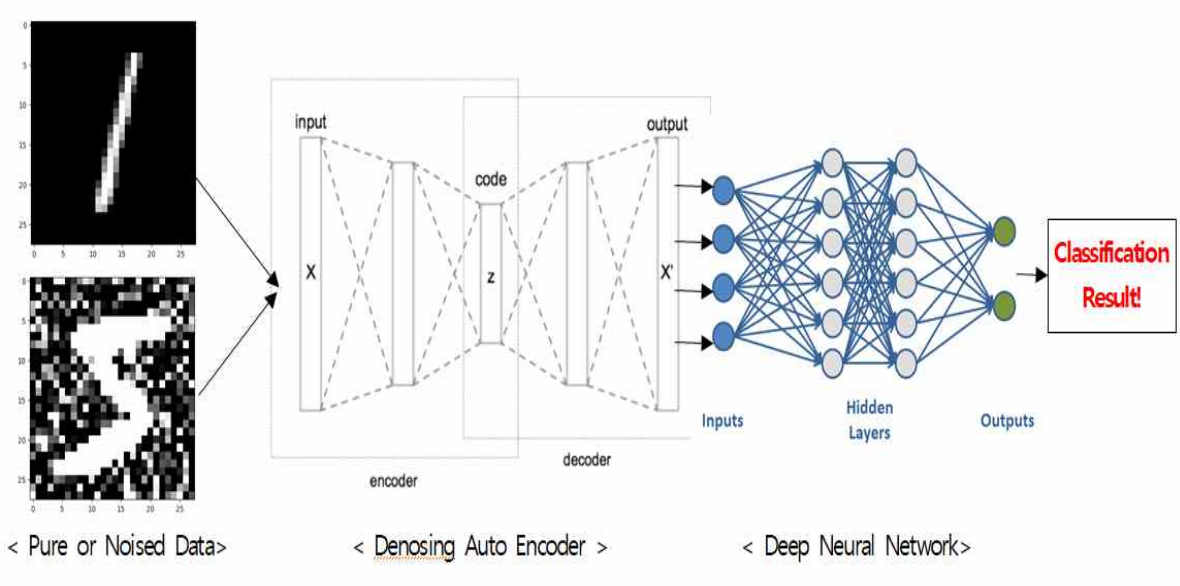


이제 숫자인식기의 전단부에 학습된 DAE를 추가한다. 모든 데이터는 DAE에 의해 노이즈가 제거된 후에, 숫자인식기의 입력으로 들어가 분류 결과를 얻게 된다. 숫자 인식기도 Keras를 이용해 구성하였으며, 각각 100개, 30개의 Perceptron을 가진 Hidden Layer(FC) 2개와 10개의 Perceptron을 가진 Output Layer(FC)로 구성된 DNN 구조를 가진다. 각 Layer의 Activation 전에는 Batch Normalization을 수행하였고, ReLU Function과 Sigmoid Function을 Activation Function으로 사용하였다. DNN 숫자 인식기의 모델 구조는 아래와 같다.



< Deep Neural Network를 이용해 구현한 숫자 인식기의 모델 구조 >

위의 그림처럼 구성된 DNN 숫자 인식기는, DAE를 거친 결과를 Input Data로 받아 0~9 사이의 숫자로 분류하는 학습을 진행한다. 학습을 위한 Cost Function 으로는 Mean Squared Error를 사용하였고, 파라미터 업데이트 방법으로는 Gradient Descent를 사용하였다.
학습은, Batch Size를 100으로 하여 Cost 감소율이 상당히 줄어드는 시점까지 하기 위해, 총 10 Epoch 동안 진행했다.



< 구현한 숫자 인식기의 전체 구조 >

- 참고 문헌

[1] DAE의 기본 개념 - 본 강의의 12주차 강의노트 중 DAE에 대한 그림 및 설명

[2] loss를 계산하는 Cross−EntropyCross−Entropy 함수 - https://wikidocs.net/3414

[3] 파라미터 업데이트 방법 중 하나인 AdaDelta 알고리즘 - http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

[4] Keras에서 Auto Encoder 를 구현하는 예시 - https://blog.keras.io/building-autoencoders-in-keras.html

[5] N-MNIST 데이터셋 - http://csc.lsu.edu/~saikat/n-mnist/

[6] 파이썬에서 scipy를 이용해 Matlab 데이터를 열기 - https://docs.scipy.org/doc/scipy/reference/tutorial/io.html

4. 프로그램 성능 평가 실험

Denosing Auto Encoder 모델의 성능은 3장의 구현 방법에서 이미지를 통해 확인해 보았으므로, 전체 숫자 인식기 모델에 대한 성능 평가 실험을 진행한다.
실험은 구현한 숫자 인식기의 분류 성능과, DAE의 효과를 검증하기 위한 실험으로 설계했다.

실험 ① : 구현한 숫자 인식기가 노이즈가 섞인 Test Data Set도 잘 분류 하는가?

- **실험 방법**

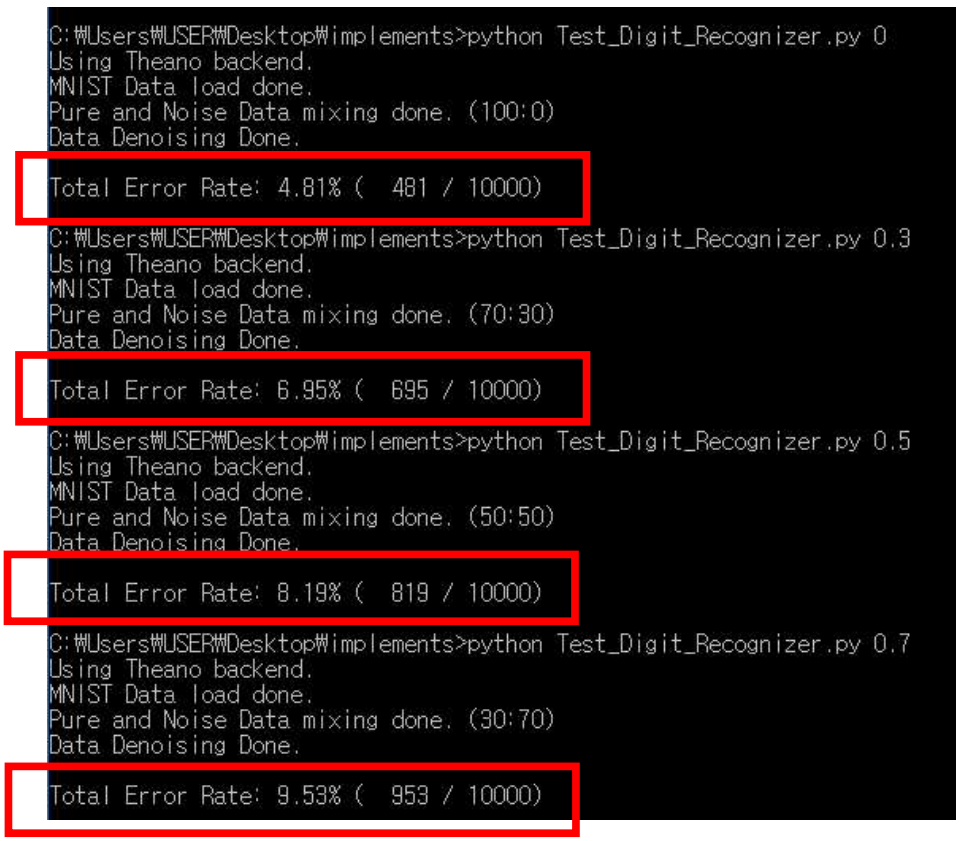
1. 구현한 숫자 인식기를 정상 Train Data Set으로만 학습 (10 Epoch)

DAE_model_param.h5(학습된 DAE 모델 파라미터) + Train_Digit_Recognizer.py (학습 프로그램)
→ Train_Digit_Recognizer_param.h5 (학습된 숫자 인식기 모델 파라미터)

2. 일부 샘플에 노이즈가 섞인 Test Data Set으로 숫자 인식기 테스트 진행

DAE_model_param.h5 + Train_Digit_Recognizer_param.h5
+ Test_Digit_Recognizer.py (테스트 프로그램)
→ 에러율 계산!

- 실험 결과 (노이즈 데이터의 비율 : 0%, 30%, 50%, 70%)



실험 ② : 원래 DAE 없이도 노이즈 데이터까지 잘 분류 되는 것은 아닌가?

- **실험 방법**

1. DAE를 사용하지 않고, 정상 Train Data Set으로 학습 (10 Epoch)

Train_Digit_Recognizer_without_DAE.py (학습 프로그램)

→ Train_Digit_Recognizer_without_DAE_param.h5 (학습된 숫자 인식기 모델 파라미터)

2. DAE를 사용하지 않고, 일부 샘플에 노이즈가 섞인 Test Data Set으로 테스트 진행

Train_Digit_Recognizer_without_DAE_param.h5

+ Test_Digit_Recognizer_without_DAE.py (테스트 프로그램)

→ 에러율 계산!

- **실험 결과 (노이즈 데이터의 비율 : 0%, 30%, 50%, 70%)**

```
C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_without_DAE.py 0
Using Theano backend.
MNIST Data load done.
Pure and Noise Data mixing done. (100:0)

Total Error Rate: 3.39% (  339 / 10000)

C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_without_DAE.py 0.3
Using Theano backend.
MNIST Data load done.
Pure and Noise Data mixing done. (70:30)

Total Error Rate: 29.55% ( 2955 / 10000)

C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_without_DAE.py 0.5
Using Theano backend.
MNIST Data load done.
Pure and Noise Data mixing done. (50:50)

Total Error Rate: 46.98% ( 4698 / 10000)

C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_without_DAE.py 0.7
Using Theano backend.
MNIST Data load done.
Pure and Noise Data mixing done. (30:70)

Total Error Rate: 64.54% ( 6454 / 10000)
```

실험 ③ : DAE 없이, 애초에 정상 데이터와 노이즈가 섞인 데이터를 모두 학습하면 안되나?

- **실험 방법**

1. DAE를 사용하지 않고, 정상 Data Set + 노이즈 Data Set을 반씩 섞어서 학습 (20 Epoch)

Train_Digit_Recognizer_without_DAE_by_mixed_data.py (학습 프로그램)

→ Train_Digit_Recognizer_without_DAE_by_mixed_data_param.h5 (학습된 숫자 인식기 모델 파라미터)

2. DAE를 사용하지 않고, 일부 샘플에 노이즈가 섞인 Test Data Set으로 테스트 진행

Train_Digit_Recognizer_without_DAE_by_mixed_data_param.h5

+ Test_Digit_Recognizer_without_DAE_by_mixed_data.py (테스트 프로그램)

→ 에러율 계산!

- **실험 결과 (노이즈 데이터의 비율 : 0%, 30%, 50%, 70%)**

```
C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_without_DAE_by_mixed_data.py 0
Using Theano backend.
MNIST Data load done.
Pure and Noise Data mixing done. (100:0)

Total Error Rate: 3.91% (  391 / 10000)

C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_without_DAE_by_mixed_data.py 0.3
Using Theano backend.
MNIST Data load done.
Pure and Noise Data mixing done. (70:30)

Total Error Rate: 25.27% ( 2527 / 10000)

C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_without_DAE_by_mixed_data.py 0.5
Using Theano backend.
MNIST Data load done.
Pure and Noise Data mixing done. (50:50)

Total Error Rate: 39.99% ( 3999 / 10000)

C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_without_DAE_by_mixed_data.py 0.7
Using Theano backend.
MNIST Data load done.
Pure and Noise Data mixing done. (30:70)

Total Error Rate: 54.30% ( 5430 / 10000)
```

실험 ④ : 구현한 숫자 인식기가 다른 방식의 노이즈가 섞인 Data Set도 잘 분류 하는가?

- **실험 방법**

1. 실험 ①, ②와 같은 방식으로 실험하되, Test Data Set을 N-MNIST 데이터로 바꾼다.

Test_Digit_Recognizer_tested_N-MNIST.py (학습 프로그램)

→ 에러율 계산!

- N-MNIST 데이터란?

MNIST 데이터셋에 아래 3가지 종류의 노이즈를 각각 추가해 만든 데이터셋이다.

(1) Additive White Gaussian Noise (AWGN)

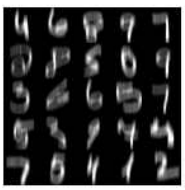
(2) Motion Blur

(3) A Combination of Additive White Gaussian Noise and Reduced Contrast

본 데이터는 아래 페이지에서 다운로드 할 수 있으며, MATLAB 데이터로 인코딩 되어있어서 Python의 scipy.io 라이브러리를 통해 불러와서 원하는 Numpy Array 형태로 정리하였다. 해당 함수는 module_load_data.py 모듈에 정의되어 있다.

Sample images from the n-MNIST dataset:

n-MNIST with Additive White Gaussian Noise (AWGN)



n-MNIST with Motion Blur

n-MNIST with reduced contrast and AWGN

< 관련 페이지 - http://csc.lsu.edu/~saikat/n-mnist/ >

- **실험 결과**

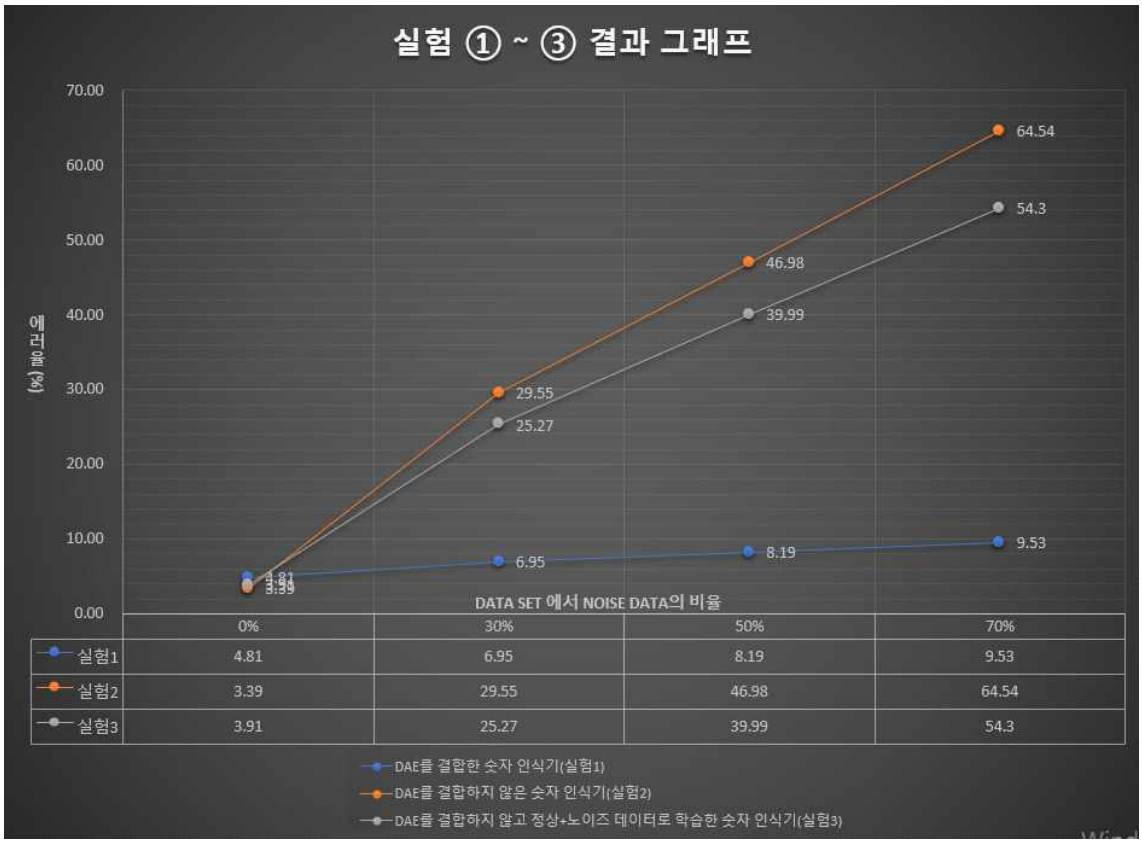
```
C:₩Users₩USER₩Desktop₩implements>python Test_Digit_Recognizer_tested_N-MNIST.py
Using Theano backend.
N-MNIST Data load done.

---Test without DAE---
AWGN -> Total Error Rate: 28.27% ( 2827 / 10000)
 MB  -> Total Error Rate: 5.13% (  513 / 10000)
 RC  -> Total Error Rate: 42.53% ( 4253 / 10000)

Denoising Auto Encoder Model is loaded.
AWGN(Additive White Gaussian Noise) Data Set Denoising Done.
Motion Blur Data Set Denoising Done.
Reduced Contrast and AWGN Data Set Denoising Done.

---Test with DAE---
AWGN -> Total Error Rate: 12.86% ( 1286 / 10000)
 MB  -> Total Error Rate: 7.69% (  769 / 10000)
 RC  -> Total Error Rate: 20.00% ( 2000 / 10000)
```

5. 실험 결과 분석 및 개선 방안



최종 구현한 모델을 사용한 실험 ①의 결과, 노이즈 데이터의 비율이 증가함에 따라 에러율이 조금씩 증가하는 추세를 보이지만, 노이즈 데이터의 비율이 70% 까지 올라가더라도 10% 이하의 에러율을 보이며 노이즈에 강한 숫자 인식기로 작동함을 확인할 수 있다.

DAE를 사용하지 않은 실험 ②의 경우, 노이즈 비율이 증가하는 만큼 에러율도 증가하는 추세를 보인다. 이를 통해, DAE를 사용하지 않으면 노이즈에 매우 취약한 숫자 인식기가 학습된다는 사실을 알 수 있다.

마지막으로, DAE 없이 노이즈 데이터까지 학습시킨 실험 ③의 경우, 노이즈 데이터를 학습하지 못한 실험 ② 보다는 낮은 에러율을 보였다. 하지만 노이즈 데이터 비율이 증가함에 따라 에러율도 크게 증가하였고, DAE를 사용한 모델에 비해 현저히 높은 에러율을 보였다.

새로운 형태의 노이즈에도 강한지 알아보기 위한 실험 ④의 결과는 위와 같았다. AWGN과 AWGN+Reduced Contrast에는 DAE를 추가해 구현한 본 프로그램을 이용할 때 훨씬 좋은 인식률을 보여줌을 확인할 수 있다.
하지만 Motion Blur 노이즈에 대해선 오히려 에러율이 약간 증가하는 양상을 보였는데, 이는 Motion Blur의 노이즈 형태가 AWGN과 상당히 다르고 AWGN에 비해 비교적 원본 이미지의 형태가 잘 보존되기 때문으로 보인다. 하지만 에러율이 증가했음에도 불구하고, 10% 이하의 에러율을 보이며 여전히 높은 인식률을 보인다고 할 수 있다.
만약 Motion Blur 노이즈가 섞인 Data Set의 Train Data List를 로드하여 DAE 학습시에 같이 학습해준다면, 에러율이 낮아지게끔 개선될 수 있을 것이다.

6. 결론

위의 실험결과들을 통해서 DAE를 결합하여 숫자 인식기를 구현할 경우, 노이즈가 섞인 데이터에 대해 DAE가 없는 숫자 인식기보다 높은 인식률을 보인다는 사실을 알 수 있다. 하지만 기존의 숫자 인식기에 DAE 모델을 추가해야 한다는 점에서, 추가적인 연산이 필요하기 때문에 본 프로그램은 노이즈가 빈번히 발생하는 환경에서만 사용하는 것이 타당할 것이다. 또, 현재 DAE를 구성한 모델이 다소 복잡한 형태를 띠고 있는데(각 Layer의 Filter 개수가 32개), 필터의 개수를 줄이거나, 레이어를 축소해보며 결과를 비교해본다면, 보다 적은 연산으로 최적의 DAE를 구성해낼 수 있을 것이다.