MNIST 데이터를 이용한 필기체 글자 인식 응용 프로그램 작성

201601819 박 슬기

1. 서론

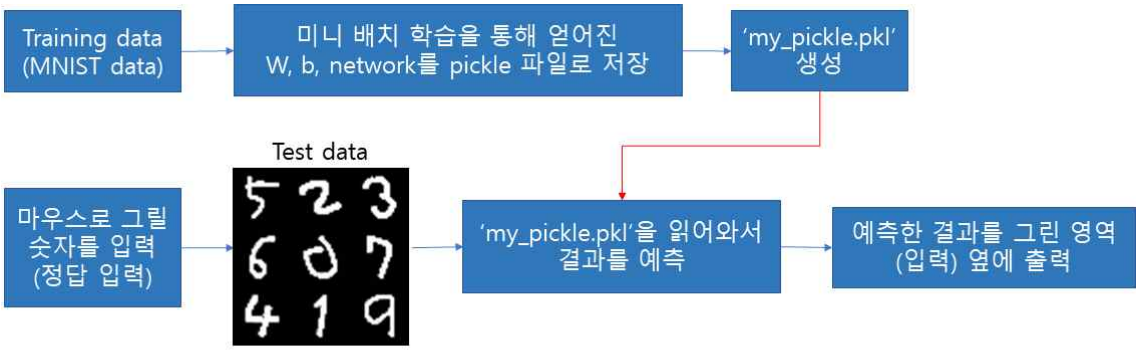
MNIST 데이터를 이용한 필기체 글자 인식 프로그램은 많은 사람들이 여러 언어로 구현을 해놓았다. 거의 완벽에 가까운 인식률을 자랑하는 프로그램들도 있고, 돌팔이 의사가 떠오를 만큼 저조한 인식률을 가진 프로그램들도 있다.

본 보고서에서는 마우스로 입력한 필기체 글자를 MNIST 데이터와 비슷하게 만들어 입력 데이터를 생성하고, Two-Layer Network에 입력 데이터를 넣어 예측 결과를 출력하는 필기체 글자 인식 응용 프로그램을 Python으로 구현하였다. MNIST 데이터들의 모양(생김새)을(를) 파악하고 그와 비슷하게 필기체 글자를 입력하면 어느 정도 올바르게 예측된 결과를 출력하도록 하였고, 본인을 제외한 여러 사람들의 필기체 글자로 test를 함으로써 각 숫자별로 다양한 모양을 입력하였을 때 몇 개의 숫자를 제외하고는 90%에 가까운 인식률을 보이도록 구현하였다.

2. 본론

[그림 1]은(는) 구현한 필기체 글자 인식 프로그램의 순서도이다. MNIST 데이터를 (normalize=True, one_hot_label=True)로 업로드 하여 입력으로 하고, hidden_size = 500, batch_size = 100으로 하는 미니 배치 학습을 통해 얻은 가중치(W), bias(b), network 구조를 pickle 파일로 저장하여 'my_pickle.pkl'을 생성한다. 이 미니 배치 학습의 정확도(accuracy)는 약 97% 정도로 낮지 않은 편이다.

Test 단계에서는 인식할 숫자를 idle 창에 입력하고 마우스로 글자를 그리게 된다. 이 때 마우스로 그린 글자가 test data가 되고 idle 창에 입력한 값이 원래 내가 생각한 정답이 된다. test data를 입력 후 'space bar'를 누르면 그린 영역을 사각형으로 받아서 가로와 세로의 비율을 비슷하게 맞춘 후 크기를 20x20으로 resize하고, 이 데이터를 MNIST 데이터와 비슷한 형태로 만들기 위해 크기가 28x28인 배열의 정 가운데에 resize한 데이터를 넣어 입력 데이터를 만든다. 입력 데이터를 얻은 후 미니 배치를 통해 얻은 pickle 파일을 읽어와서 정답일 확률이 가장 높은 원소의 인덱스를 얻어오게 되는데 이것이 결과 예측이다. 이 예측한 결과는 그린 영역의 우측에 출력되며, 'v'를 누르면 처음에 내가 idle 창에 입력한 값과 예측한 결과 값을 비교하여 몇 번 중에 몇 번 맞추었고, 맞출 확률이 얼마나 되는지를 idle 창에 출력한다.



[그림 1] 프로그램 구조 및 실행 순서

2.1 미니 배치 학습을 통해 얻어진 W, b, network를 pickle 파일로 저장

[표 1]을(를) 통해 hidden_size를 설정하였는데 5번씩 실행해본 결과를 보면 average 값이 크게 차이는 나지 않지만 조금이라도 높은 인식률을 위해 가장 높은 값으로 설정하였다.

hidden_size	1차	2차	3차	4차	5차	average
500	0.9767	0.9772	0.9774	0.9762	0.9771	0.97692
200	0.9742	0.9745	0.9758	0.976	0.9758	0.97526
100	0.9758	0.9748	0.9745	0.9748	0.9731	0.9746
50	0.9715	0.9701	0.97	0.9704	0.9687	0.97014

[표 1] hidden_size에 따른 test_accuracy (batch_size=100, learning_rate=0.1)

[표 2]를 통해 learning rate를 설정하였는데 5번씩 실행해본 결과를 보면 average 값이 [표 1]의 차이보다 훨씬 큰 차이를 보이는 것을 알 수 있다. 이러한 결과를 바탕으로 learning rate를 훨씬 더 큰 값인 0.1로 설정하였다.

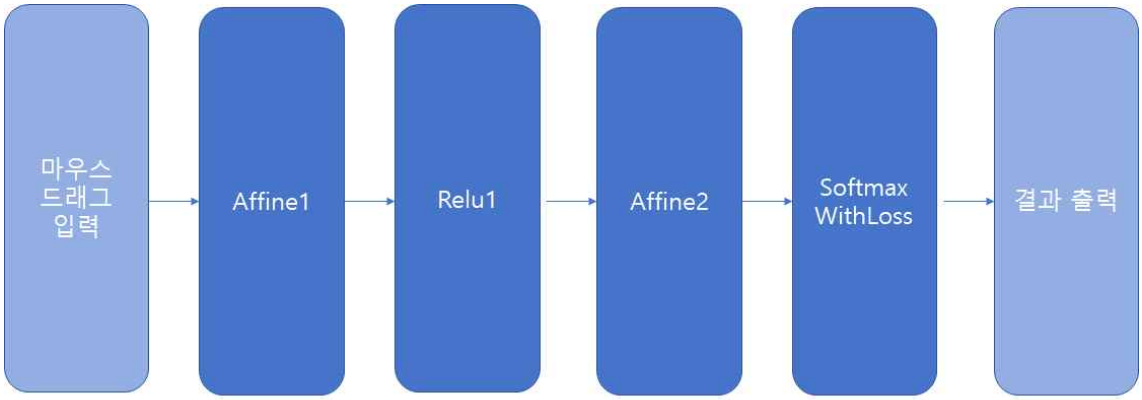
learning rate	1차	2차	3차	4차	5차	average
0.1	0.9767	0.9772	0.9774	0.9762	0.9771	0.97692
0.01	0.9282	0.9278	0.9277	0.928	0.9278	0.9279

[표 2] learning_rate에 따른 test_accuracy (batch_size=100, hidden_size=500)

이렇게 설정된 hidden_size와 learning rate를 적용하여 미니 배치를 실행하였고, 이를 통해 얻어진 가중치(W)와 bias(b), network를 'my_pickle.pkl'라는 pickle 파일에 저장하였다.

2.2 'my_pickle.pkl' 읽어 와서 결과를 예측

[그림 2]는 미니 배치 결과로 생성된 'my_pickle.pkl' 파일의 network로 [Affine1 - Relu1]가 한 층, [Affine2 - SoftmaxWithLoss]가 한 층을 구성하는 Two-Layer Network 이다. 마우스를 드래그하여 dst에 필기체 글자를 입력하면 [Affine1 - Relu1]계층을 거쳐서 나온 결과를 다시 [Affine2 - SoftmaxWithLoss]계층에 넣어서 최종 예측 결과를 출력하는 구조이다. Training 과정(미니 배치)에서 이 네트워크의 각 층을 거칠 때마다 매개변수 [W1, b1], [W2, b2]를 각각 갱신하고 네트워크 구조까지 함께 pickle 파일에 저장했으므로 Test 과정에서는 저장된 매개변수와 네트워크 구조를 그대로 읽어 와서 확률이 가장 높은 숫자의 인덱스를 결과로 출력한다.



[그림 2] network 구조

3. 구현 및 결과

윈도우즈 10 플랫폼의 노트북에서 파이썬(3.6.1)을 이용하여 구현하였으며, Training data로 MNIST 데이터를 (normalize=True, one_hot_label=True)로 업로드 하여 사용하였고, Two-Layer Network는 '밑바닥부터 시작하는 딥러닝'의 5장(오차역전파법)에 있는 예제 중에서 two_layer_net.py를 사용하였으며, 4장(신경망 학습)의 train_neuralnet.py를 이용하여 training 하였다.

사용자가 입력할 숫자를 먼저 idle 창에 입력을 하고 마우스로 필기체 글자를 입력한 다음 'space bar'를 누르면 예상 결과를 입력한 영역(그린 영역) 우측에 노란색으로 출력하게 되고, 100번 이내로 입력 후 'v'를 누르게 되면 입력 횟수와 정답을 맞춘 횟수, 정답률(인식률)을 idle 창에 출력하고 대기상태가 되도록 하였다.

내가 입력한 값과 예상 결과가 다른 경우 자동으로 실험 날짜를 이름으로 하는 폴더를 생성하여 그 안에 입력 데이터(크기: 28x28)를 이미지로 저장하도록 하였고, 입력한 값과 예상 결과가 같은 경우에도 저장을 원할 때 'r'을 눌러서 저장할 수 있도록 하였다.

3.1 인식률을 높이기 위한 여러 실험 및 개선

1. training과 test에 교재 3장의 예제 two_layer_net.py를 network로 사용하였더니 training 과정에서의 인식률이 94%, 실제 필기체 글자 인식률은 50% 미만으로 나타났다.
   → 5장의 예제 two_layer_net.py로 network를 교체 후 training과정에서의 인식률이 97%, 실제 필기체 글자 인식률이 50~60%로 높아졌다.
   (3, 4, 5, 7, 8, 9의 인식률은 다른 숫자들에 비해 낮았다.)

2. 필기체 글자 인식률을 더 높이기 위해 dropout을 사용해 보았지만 인식률이 더 낮아지고 출력도 2개가 나왔다.
   → 1.에서 개선한 부분으로 다시 돌려놓았다.

3. 마우스 왼쪽 버튼을 누르고 움직일 때 생기는 원의 크기를 20에서 18, 15, 12, 10, 9로 여러 번 변경하였다.
   → 적당한 크기인 12로 원의 크기를 조정하였다.

4. training data는 normalize = True로 training하고, test data는 normalize = False로 사용하고 있었던 것을 알고 test data를 astype(float32)로 한 후 255로 나누어 normalize = True와 같은 효과를 내주었다.
   → 인식률이 낮았던 3, 5, 8, 9의 인식률이 높아졌다.

3.2 최종 실험 결과

0부터 9까지의 숫자를 각각 100번씩 입력하여 예측 결과를 확인하였다. [표 3]은(는) 각 숫자별 입력에 따른 실험 결과이다. [표 3]의 결과를 좀 더 한눈에 보기 쉽도록 하기 위해 [그래프 1]로 나타내었다. [그래프 1]을(를) 살펴보면 4와 7의 인식률이 다른 숫자들에 비해 저조하다는 것을 알 수 있다.

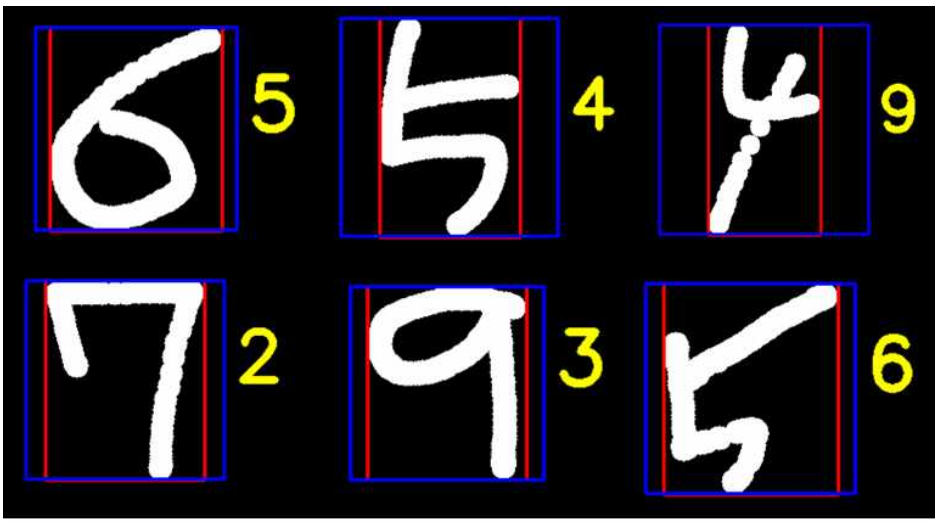
인식 숫자	입력 횟수(회)	맞춘 횟수(회)	인식률(%)
0	100	96	96
1	100	88	88
2	100	91	91
3	100	93	93
4	100	68	68
5	100	76	76
6	100	75	75
7	100	56	56
8	100	94	94
9	100	75	75

[표 3] 실험 결과

[그래프 1] 실험 결과

[그림 3]은 여러 인식 결과들 중에서 잘못 인식한 결과들만 몇 개 가져온 것이다. 이 결과들을 살펴보면 사람의 눈으로 판단했을 때 사용자의 의도와 동일한 판단을 할 수 있는 데이

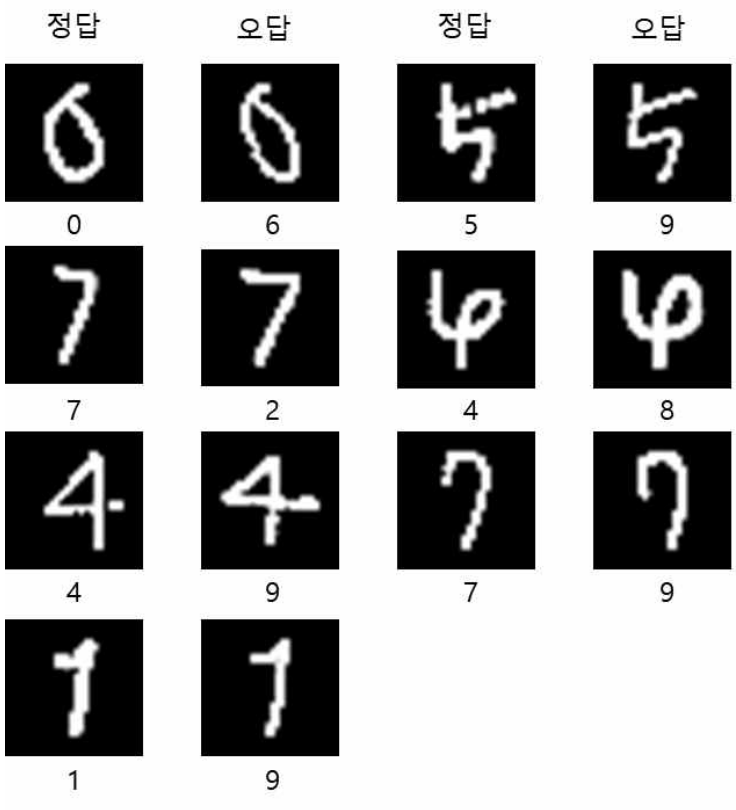
터들임에도 불구하고 잘못된 결과를 출력하였다. 이는 training 단계에서 사용한 MNIST 데이터들이 좀 더 다양했더라면 충분히 올바른 예측을 할 수 있는 부분이지만 현재 상황으로는 해당 데이터가 조금 부족하여 이런 결과를 얻게 되었다고 생각한다.



[그림 3] 잘못 인식한 결과

3.3 비슷한 모양이지만 다른 결과들

3.2에서 저조한 인식률을 보였던 숫자들 중에서 서로 비슷한 모양이지만 예측 결과가 정답인 것과 오답인 것들을 살펴보겠다. [그림 4]를 보면 0, 5, 7, 4, 1을 각각 6, 9, 2, 8, 9, 9, 9로 인식하였는데 그냥 사람의 눈으로 보면 정말 비슷한 모양임에도 예측 결과가 다르게 나왔다.



[그림 4] 비슷한 모양의 정답과 오답

3.4 느낀 점

책의 예제를 실행시켜보니 MNIST 데이터를 training과 test에 모두 사용하면 training accuracy와 test accuracy가 97%나 되었지만 실제로 training에만 MNIST 데이터를 사용하고 필기체 글자를 test에 사용해보니 인식률이 50%도 되지 않았다. 이런 실망스러운 결과를 두고 고민도 해보고 여러 방법도 찾아보면서 평소보다 더 많은 생각들을 하게 되었다. 프로그램 하나를 만들기 위해 엄청난 시간을 투자하고, 여러 시행착오를 거쳐 어느 정도 만족스러운 결과물을 만들어 내는 과정에서 완벽한 인식률을 보이지는 못했지만 생각의 방향과 깊이가 조금 더 발전한 것 같아 만족스럽게 프로젝트를 마무리하였다.

training 데이터가 조금 부족하여 더 많은 종류의 필기체 글자를 정확히 인식하지는 못하였지만 제출 후에 training 데이터로 MNIST 데이터에 여러 사람의 필기체 글자를 추가하여 training 시키고 거의 완벽하게 인식하는 프로그램을 꼭 만들고 싶다.

4. 참고 문헌

[1] 밑바닥부터 시작하는 딥러닝(사이토 고키)

[2] https://opencv-python.readthedocs.io/en/latest/doc/03.drawShape/drawShape.html

[3] http://w3devlabs.net/wp/?p=16810