Machine Learning

Assignment #2



수업 일시 : 금 5, 6

담당 교수 : 이 혁 준 교수님

학 과 : 컴퓨터 공학과

학 번 : 2013722095

이 름 : 최 재 은

1. 제목

MNIST, Recognition of handwrite number

by convolution and pooling

1. 목적

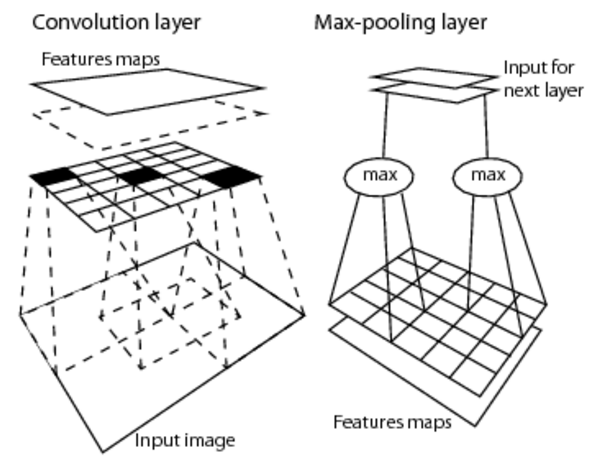
필기체 숫자 0~9를 인식하는 모델인 MNIST 모델을 구현해 본다. 이를 위해 Convolution Neural Network가 어떻게 동작하는지를 이해하고, 또 이를 통해 Convolution, Pooling을 거친 데이터가 어떻게 출력되는지, 정확도를 더 높일 수 있는 방법은 무엇이 있는지를 찾아보고 적용해본다.

1. 원리

Convolution이란 주로 신호 및 시스템에서 주로 사용하는 operation으로써 어떤 필터를 사용하여 주어진 image의 feature를 뽑아낸다. CNN(Convolution Neural Network)란 Convolution을 이용한 network로써 임의의 크기를 갖는 필터를 이미지 위로 이동시키며(정확하게는 Data에 대해 matching하면서) 필터의 내부에 해당되는 원본 이미지의 영역을 convolution(feature 뽑기)한다. 이는 사람의 눈이 물체를 인식하는 과정을 모방하여 만들어 냈으며 실제 동작 원리 또한 사람의 사물 인식을 모방하여 설계되었다.

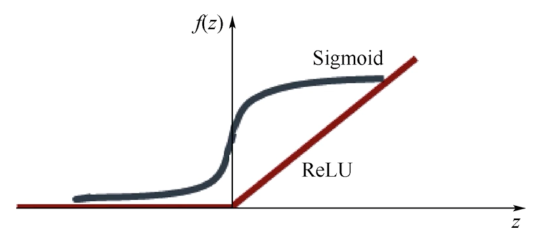
Convolution하고자 하는 이미지 크기가 (N x N)이고, 필터의 size가 (K x K)이며 stride(이동 보폭)을 s라고 할 때 convolution 결과는 한 변의 크기가 (((N - K)/s) +1)인 이미지가 된다. 이 때에 원본의 말단 부분에서 K보다 작은 영역의 feature까지 뽑아내기 위해서 padding이라는 기술을 적용할 수도 있는데, 이는 K보다 작은 영역에 0을 만들어 줌으로써 좀더 sensitive한 feature를 뽑을 수 있도록 도와주기도 한다.(사실 이는 뽑아내고자 하는 모델의 특징에 따라 다르기 때문에 직접 해봐야 안다…)

Pooling은 filter size영역 내의 값들을 하나의 value로 압축시키는 동작을 수행한다. 이 때에는 영역 내의 평균이나, 최소값, 최대값 등을 적용시키는데 최근 들어서는 최대값을 쓰는 MAX Pooling을 적용한다. Pooling 또한 convolution과 같은 원리로 결과 크기를 계산할 수 있으며, convolution으로 뽑아낸 feature를 유지하는 선에서 data의 크기를 압축하는 역할을 한다.



CNN의 주요 원리는 Pre-processing이 실제 동작에 있어 영향이 크기 때문에 위와 같은 Pre-processing을 통해 원본 Data의 특징을 미리 잘 뽑아내어 적용하자는 것이다. 이렇게 Pre-processing을 거친 Input Data들은 기존의 Neural Network와 같이 hidden layer에 들어가게 된다. 이 때의 hidden layer는 fully connected layer로써 2~3개의 layer를 갖는다. 그 이후에 나오게 되는 data는 SoftMax Layer에 들어가게 되어 정해진 class들에 대한 각각의 확률 값을 반환하게 되는데, 이를 통해서 컴퓨터는 입력된 input data가 어떤 물체인지를 인식하게 된다.

CNN에서는 convolution layer를 거친 data가 활성화 함수를 거치게 한다. 각 data들을 결합하여 하나의 정보로 만들어 주는 것인데 입력 정보의 결합 값을 일정 범위의 값으로 전환해 준다. 일정 값을 넘어야 성공, True가 되기 때문에 활성화 함수로도 불린다. 활성화 함수로써 주로 sigmoid 함수를 사용하며 예로 Logistic함수와 Hyperbolic tan함수를 사용했으나, 이들은 back-propagation에서 layer가 깊어질수록 특징이 흐릿해지는 Vanishing gradient Problem을 만들어 냈다. 기울기를 감소시키면서 Optimum value를 찾아내야 하는데, 위의 함수들은 기울기가 발산, 소실 되는 문제를 가져왔었다. 기울기 발산의 문제는 clipping이라는 기술을 적용하여 발산 한계를 둠으로써 해결할 수 있었으나 기울기 소실에 대한 문제는 항상 큰 과제였다. 이 때문에 learning rate의 조절이 큰 영향을 끼쳤었다. 이러한 문제를 해결하기 위해 등장한 함수가 ReLu 활성함수이다.



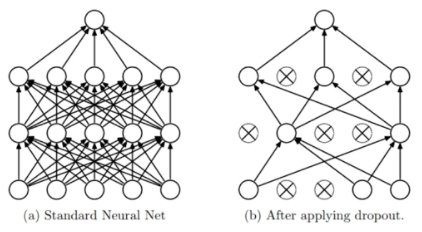
위의 그림과 같이 0보다 작은 값에 대해서는 0을 적용하고, 0보다 큰 값에서는 해당 값을 그대로 사용하는 방법이다.

위에서는 gradient Vanishing을 해결함으로써 잘 back – propagation할 수 있도록 하였다. Performance를 높이는 또 하나의 방법으로 기울기를 찾아가는 방법을 개선하는 것이 있다. 요새는 기존의 Gradient Descent Optimizer 대신 AdamOptimizer를 사용한다고 한다.

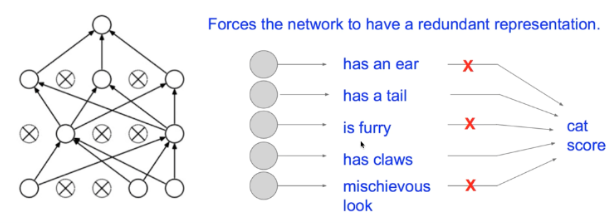
모델의 Accuracy를 높일 수 있는 또 하나의 방법으로 초기의 Weight를 잘 설정해 주는 것이 있다. 처음에 대충 때려 맞춘 값이 정답에 가깝다면 아무래도 수행 속도나 performance가 더 좋기 때문에 초기화를 잘 하는 것이 나름 중요하다고 한다. Xavier, He, MSRA, LSUV 등 여러 initializer가 있으나 He가 가장 단순하고 성능이 좋다고 한다.

다시 본론으로 돌아가 Neural Network는 인간의 뇌 신경망을 모방하여 만든 인공신경망이다. 각 hidden layer끼리 정보를 주고 Back-propagation함으로써 모델을 형성하고 동작하는데, 얼핏 보기에는 layer가 깊어질수록 더 정확히 볼 수 있기 때문에 성능이 좋아질 것처럼 보인다. 그러나 Fully connected 방식에선 오히려 이런 방식이 Input Train Data에 대하여 Overfitting을 발생시키기 때문에 Test함에 있어 Accuracy가 낮아지는 결과를 가져온다. 즉 훈련한 것에 대해서만 성능이 좋고 다재 다능하지 않다는 뜻이 돼버린다. 이러한 overfitting 문제를 해결하기 위해서 DropOut이라는 기술을 적용할 수 있다.

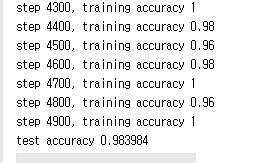
이는 전체 weight를 계산에 참여시키는 것이 아니라 Layer에 포함된 weight 중 일부만을 참여시키는 것이다. 딥 러닝을 공부할 때에 오히려 feature에 대한 정보가 소실돼 버리는 것이니 정확도가 엉망일 것이라 생각했으나 오히려 성능이 더 좋아지는 결과를 보였다.



데이터를 다음 layer로 전달할 때, 난수를 사용하여 일부 neuron을 0으로 만드는 방법을 사용한다. 이 과정에 딱 어울리는 고사로 ‘사공이 많으면 배가 산으로 간다’라는 말이 있다. 오히려 너무 많은 점을 다 따지면 이도 저도 아닌 값이 되는 것이다. 정확한 특징을 알고만 있다면, 몇몇의 weight만으로도 충분한 결과를 뽑아낼 수 있고, 이러한 과정을 반복적으로 수행한다면 오히려 균형 잡힌 훌륭한 결과를 뽑을 수 있다는 이론이다. 물론 실제로도 잘 돌아간다. 아래의 사진은 딥 러닝을 공부할 때 Dropout의이해를 크게 도왔던 강의 자료임에 첨부하였다.



1. 결과



* 위에서 확인할 수 있듯이 5000 epoch를 진행한다. 1 epoch당 50개의 data를 가져다 쓰므로 250000개의 data를 통해서 훈련하고 모델을 형성하는 것이다.
* 후반부로 갈수록 중간 중간에 1의 정확도를 보이기도 한다.
* 테스트 정확도는 0.983984가 나왔다.
* 사실 요즘 나오는 MNIST는 대부분 약 98%의 정확도를 보인다고 한다.

1. 고찰

이번 과제에서 어떤 convolution과 pooling 함수에서 padding을 사용할 것인지를 입력 받을 수 있도록 함수가 구현되어 있었으나, 앞의 함수의 입력 인자의 수나 20 page에서의 전체적인 형태를 보았을 때 flatModel에 들어가는 data의 shape가 4\*4\*32형태 였기 때문에 padding을 Valid로 두고(미 사용) 전체 입출력을 조정하였다. Python을 공부해본 적이 없어서 Placeholder 뭔지도 모르고 있다가 오늘에서 data를 담는 그릇 같은 역할을 함을 알게 되었다. 다음에 수행할 Data를 뽑아오는 Nextbatch 함수에서는 data의 수를 list로 만들고 그걸 무작위로 섞은 뒤에 앞에서부터 50개의 Index를 뽑아 list로 다시 만들어 주었다. 그 Index 값을 통해서 Data와 그에 matching 되는 label을 뽑아서 각각 array로 만들어 반환해 주도록 하였다. 무작위로 섞어 주는 과정에서 np.random.permutation()과 np.random.shuffle()에서 고민을 많이 했었다. 솔직히 말하면 두 함수는 동일하게 무작위로 섞어주는 역할만 하는데 왜 permutation을 쓰면 200 step만에 accuracy가 1을 찍는지 모르겠다. 왠지 너무 빨리 정확도가 1이 되는게 overfit인가 싶어 np.random.shuffle()을 사용하였다. 또 정확도를 높이기 위해서 Adam initializer를 사용하였고 Convolution이 끝나면 ReLu Activation Function을 적용하였다. 위의 결과에서도 언급했다시피 왠만한 MNIST 모델들이 대부분 약 98%의 정확도를 보인다고 한다. Xavier Initializer, AdamOptimizer 등을 적용하게 되면 99%를 넘기도 한다고 한다. 그럼에도 결국 중요한 것은 얼마나 정확하고 다양한 Data Set을 잘 마련하는가, 이를 통해 얼마나 잘 모델을 훈련시키는지 인 듯싶다.

\*참고

<http://pythonkim.tistory.com/52>

<https://ko.wikipedia.org>