9. CNN

P330~371

CNN은 grid-like topology이다. 예시는 time-series data를 포함한다. 그것은 regular time interval에적용되는 1d grid로 여겨지고 2d grid로 여겨지는 image data를 포함한다. CNN은 아주 잘 된다. CNN은 수학연산 모델인 convolution을 수행한다. 선형 operation의 일종이다.

9.1 Convolution Operation

Cn은 실수가 올려진 두 함수에서의 연산이다. Convolution의 정의를 동기부여하기 위하여 두 함수의 예를 시작하겠다.

레이저를 쏘는 공간에 있다고 생각해보자. 우리의 레이저 센서는 단일 아웃풋 x(t)를 제공한다. 우주선에서의 시간 t에 대한 위치이다. X와 t는 둘다 실수이다.

이제 레이저 센서에 노이즈가 있다고 생각해보자. 우주선의 위치의 노이즈가 덜한 예측값을 얻기 위해, 몇몇의 측정을 평균낸다. 가중치를 준 평균으ㅡㄹ 사용한다. W(a)이고 a는 측정의 나이?이다. 만약 매 순간 가중치를 준 평균 연산을 적용한다면, 우리는 새로운 함수 s를 갖는다. 이것은 우주선의 위치의 예측값을 부드럽게 한다. (적분, 수식 9.1)

이것을 convolution이라 불고 전형적으로 \* 를 사용한다. S(t) = (x\*w)(t) (9.2)

이 예제에서, w는 확률 density 분포 함수가 되어야 하거나 아웃풋이 가중치 평균이 되어선 안된다. 또한 w는 모든 negative arguments들에게 0이 될 필요가 있다. 또는 그것은 미래에 보게 될 것이다. 아마 우리가 다룰 주제 밖이다. 이러한 제한들은 하지만 우리의 예쩨에 특정하다. 일반적으로는 cn은 위에서 적분이 적용된 것처럼 아무 함수에나 정의 되며 가중치된 평균들을 갖는 하에 다른 목적을 위해 사용된다.

Cnn 용어에서, 첫번쨰 argument(예제에서 함수 x)는 종종 input으로써 참조되며 두번쨰 argument(함수 w)는 커널로 여겨진다. Output는 떄떄로 feature map으로 여겨진다.

우리의 예제에서, 레지어 아이디어는 실제적이지 않다. 보통 우리는 컴퓨터의 데이터로 작업하며 시간은 이산적이다. 그리고 우리의 센서는 regular intervals에서 제공된다. 우리의 예제에서, 우리의 레이저가 초당 측정 값을 제공한다는 가정은 보다 실제적이다. 시간 인덱스 t는 오직 integer 값만 제공할 수 있다. 만약 x와 w가 오직 integer t에서 정의되었다고 가정한다면 우리는 이산 cn을 정의할 수 있다.

적분이 시그마로 (9.3)

머신러닝 어플리케이션에서, 인풋은 보통 데이터의 multidimensional 어레이이다. 그리고 커널은 학습 알고리즘에의해 적용된 파라미터들의 multidimensional 어레이이다ㅏ. ㅇ우리는 이러한 멀티디멘져널 어레이를 텐서로써 참조한다. 인푸ㅜㅅ과 커널의 각 엘레멘트는 반드시 명시적으로 분리되어 저장되어야 하기 떄문에 , 위는 보통 이러한 함수들은 어디에서나 0 라고 가정했으며 하지만 우리가 값을 저장해야하는 것을 위한 포인트의 유한한 집합은 아니다. 이것은 실제로 우리는 어레이 요소들의 유한한 수를 더함으로써 무한한 덧셈을 구현할 수 있다.

마지막으로, 우리는 종종 그 시간에 하나 이상의 cn을 사용한다. 예를들어, 우리 인풋으로써 2차원 이미지 I를 사용한다면 우리는 아마 2차원 커널 K를 사용해야한다.

식 9.4

Convolution은 교환이 가능하기 때문에 9.5도 가능하다.

보통 두번째 수식이 머신러닝 라이브러리에서 구현되기에 쉽다. 왜냐하면 m과 n의 가능한 값의 범위에서 varation이 덜하기 때문이다?

컨벌루션의 교환 값은 인풋과 관련되 커널을 flip해왔기 때문에 일어난다. M이 증가하는 곳에서. 인풋으로의 인덱스는 증가하지만 커널로의 인덱스는 감소한다. 커널을 flip하는 오직 하나의 이유는 교환 값을 얻기 위함이다. 교환 값이 증명을 쓰는데 유용한 반면에 그것은 보통 뉴럴 넽웤 구현에서 중요한 값은 대신에, 많은 뉴럴 넽웤 라이브러리들은 cross-correlation이라고 불리는 관련됨 함수를 구현하였다. 그것은 컨벌루션과 같지만 커널을 flipping한 것이다.

수식 (9.6)

많은 머신러닝 라이브러리들은 cross-correlation을 구현했지만 그것을 convolution으로 부른다. 이 텍스트에서 우리는 두 콜볼루션을 콘볼루션으로 따르고 flip이 되었는지 아닌지는 특정화한다. 머신러닝의 컨텍스트에서 적절한 공간에서 커널의 적절한 값은 학습할 것이다. 그래서 커널 flipping과의 컨볼루션을 기반으로한 알고리즘은 flipping없는 알고리즘에 의해 학습된 상대적인 flip된 커널을 학습할 것이다. 머신러닝에서 단일적으로 사용되는 컨볼루션은 드물다; 대신에 컨볼루션은 동시적으로 다른 함수들에 사용되고 이러한 함수들의 결합은 그 컨볼루션 연산이 그것의 커널에 flip되었는지 또는 그렇지 않았는지의 여부에 상관없이 교환하지 못한다.

그림 9.1은 커널 flipping없는 컨볼루션의 예제이다. 2D 텐서가 적용되었다.

이산 컨볼루션은 매트릭스의 곱으로 보여질 수 있다. 하지만, 그 매트릭스는 다른 entries과 동등하게 강요된 몇몇의 entries를 갖는다. 예를들어, 일변량 이산 컨볼루션에서, 매트릭스의 각 row는 하나의 엘레멘트가 이동된 row와 동등하도록 강요된다. 이것을 Toeplitz matrix라고 알려져있다. 이차원에서, doubley block circulant matrix가 컨볼루션과 대응된다. 서로서로에게 동등해야하는 몇몇의 엘레먼트들로 제한되는 이러한 것들을 추가적으로, 컨볼루션은 보통 아주 sparse matrix로 대응된다. 이것은 커널은 보통 인풋 이미지보다 아주 작기 때문이다. 매트릭스 곱과 작동하는 그리고 매트릭스 구조의 특정한 값에 의존하지 않는 아무 뉴럴 넽웤 알고리즘은 아무 더 이상의 뉴럴 넽웤으로의 변화가 필요 없이 컨볼루션과 작동해야만 한다. 전형적으로 컨볼루셔널 뉴럴 넽웤은 효과적으로 큰 인풋을 다루기 위해 특정화된 것을 사용해야만 하고 이론적인 면에서 필요하지 않는다.

9.2 Motivation

컨볼루션은 머신러닝 시스템이 향상하도록 돕는 주요 세 아이디어에 영향을 미쳤다: sparse interactions, parameter sharing, equivariant representations. 게다가 컨볼루션은 변수 크기의 인풋과 작동하기 위한 평균을 제공한다. 우리는 이 각 아이디어를 기술한다.

전형적인 뉴럴 넽웤 레이어들은 매트릭스 곱을 각 인풋 유닛과 각 아웃풋 유닛 사이의 상호작용으로 기술된 분리된 파라미터들과의 파라미터들의 매트릭스에 의해 사용된다. 이것은 모든 아웃풋 유닛들은 모든 인풋 유닛과 상호작용 한다는 것을 의미한다. 컨볼루션 네넽웤은 전형적으로 sparse interactive(또한 sparse connectivity 또는 spparse weights로 참조된다.)을 갖는다. 이것은 커널을 인풋보다 작게 만듦으로써 달성된다. 예를들어, 어떤 이미지를 프로세스할 때, 그 인풋 이미지는 천개 또는 수십만개의ㅢ 픽셀들을 갖을 것이다. 하지만 우리는 작게 검출할 수 있고 그것은 오지긱 천 또는 몇백개의 픽셀을 갖는 커널의 엣지와 같이 의미있는 특징이다. 이것은 ㅇ우리ㅣ가 더 적은 파라미터를 저장할 필요가 있다는 것을 의미한다. 메모리도 절양되고 통계적으로 효율성도 증진된다. 그것은 또한 아웃풋이 더 적은 연사을 계산한다는 것을 의미한다. 효율성에서 이러한 개선은 보통 꽤 크다. 만약 mm 인풋과 n 아웃풋이 있다면, 매트릭스 곱은 m x n 파라미터가 필요하고 그 알고리즘은 O(m x n) 런타임을 매 예제에 필요하다. 만약 우리가 아마 k를 갖는 아웃풋에 연결의 수를 제한한다면 O(k x n ) 런타임이 걸린다. 많은 실용적인 어플리케이션에서, 이것은 좋은 성능을 낸다. 머신러닝에서 k를 몇몇의 m보다 작은 크기의 순서로 유지하는 동안.

Sparse connectivity의 설명은 그래픽하게 그림 9.2와 9.3에서 나타난다. 딥 컨벌루션 네트워크에서, 딥 레이어에서 유닛들은 아마 간접적으로 더 큰 인풋의 부분과 상호작용한다. 그림 9.4에서 보여준다. 이것은 그 네트워크가 효율적으로 많은 변수들 사이의 복잡한 상호작용을 기술하도록 이러한 상호작용을 각각이 오직 sparse 상호작용을 기술하는 단일 building blocks를구성하면서 허락한다.

Parameter sharing는 하나의 모델에서 같은 파라미터를 하나 이상의 함수를 위해 사용하는 것을 참조한다. 전통적인 뉴럴 넽에서, 가중치 매트릭스의 각 요소는 정확하게 어떤 하나의 레이어의 아웃풋을 계산할 때 사용된다. 그것은 인풋의 하나의 요소로 곱해지고 다시는 방문되어지지 않는다. 파라미터 공유와 같은 동의어로써, tied wieghts라고 말 할 수 있다. 하나의 인풋에 적용된 가중치의 값은 다른 곳에서 적용된 가중치의 값과 tied되었기 때문이다. 컨볼루션 뉴럴 넽에서, 커널의 각 멤버는 인풋의 모든 위치에서 사용된다. 바운더리는 제외한다. 컨볼루션 연산에서 사용되는 파라미터 공유는 분리된 파라미터의 집합을 학습하기 보다는 오직 하나의 집합을 학습한다는 것을 의미한다.

이것은 forward propagation의 런타임에 영향을 주지 않는다. 여전히 O(k x n)이 걸린다. 하지만 이것은 k파라미터 모델의 필요한 저장 공간을 꽤 줄인다. K는 보통 m보다 적은 크기의 몇몇의 순서들을 다시 부른다. M과 n은 보통 러프하게 같은 크기이기 때문에, k는 실용적으로 m x n과 비교했을 떄 하찮은 것이다. 그러므로 컨볼루션은 아주 크게 더 dense matrix 곱 보다 메모리 또는 통계적 효율성 면에서 효과적이다. 그림9.5가 파라미터 공유에 대해서 그려준다.

컨볼루션의 경우에서, 파라미터 공유의 특정한 형태는 translation을 위한 equivariance로 불리는 어떤 값을 가지는 레이어를 야기한다. 이 함수가 equivariant하다는 말은 만약 인풋이 변했을 때 아웃풋도 같은 방법으로 변한다는 것을 의미한다. 특정하게, 만약 f(g(x)) = g(f(x))이면, 함수 f(x)가 어떤 함수 g와 equvariant하다. 컨볼루션의 경우에서, 만약 우리가 g를 input을 이동시키는 아무 함수라고 했을 때, 그 컨볼루션 함수는 g와 동등하다. 예를들어, I를 정수 좌표계에서 이미지 밝기를 주는 함수라고 하자. G를 어떤 이미지 함수를 다른 이미지 함수로 매핑하는 함수로가 하자. I` = g(I)는 이미지 함수이다. I`(x,y) = I(x-1,y). 이것은 I의 모든 픽셀을 오른쪽으로 한칸 옮기고 컨볼루션을 적용한다. 그 결과는 I`에 컨볼루션을 적용하였을 때 transformation g가 아웃풋에 적용된 것과 같은 결과이다. Time series data를 프로세싱할 때, 이것은 컨볼루션이 인풋에서 다른 특징들이 등장할 때 보여주는 timeline의 종류를 생산하는 것을 의미한다. 만약 우리가 어떤 이벤트를 제시간 뒤에 인풋에 움직인다면 정확히 같은 그것의 표현이 아웃풋에 제시간 뒤에 표현될 것이다. 이미지와 비슷하게, 컨볼루션은 인풋에서 등장하는 현재 특징들이 있는 곳에서의 2D 맵을 생성한다. 만약 그 물체를 인풋으로 움직인다면, 그것의 재표현 또한 같은 양의 아웃풋으로 이동한다. 이것은 우리가 이웃 픽셀의 작은 수의 어떤 함수르가 복수의 인풋 위치에 적용될 때 유용하나는 것을 할고 있을 떄 유용하다. 예를들어, 이미지를 프로세싱할 때, 컨볼루셔널 네트워크의 첫번째 레이어에서 엣지를 검출하는 것은 유용하다. 같은 엣지들은 많이 또는 덜 이미지에서 어디에나 존재한다. 그래서 그것은 전체 이미지에서 파라미터를 공유하는 것이 실용적이다. 어떤 경우에서, 우리는 전체 이미지를 따라 파라미터 공유를 원치 않을 것이다. 예를 들어, 만약 우리가 개인적인 얼굴의 중심이 되어 잘라진 이미지를 프로세싱하고 있다면, 우리는 아마 다른 피쳐를 다른 위치에서 추출하고 싶을 것이다.

컨볼루션은 자연적으로 다른 TRANSFORAMTION과 동등하지 않다. 이미지의 rotation이나 scale을 변형하는 transformation. 다른 메커니즘은 이러한 종류의 transformation을 다루기 위해 필요하다.

9.3 pooling p 339 9.11까지있음

컨볼루션 네트워크의 전형적인 레이어는 세 단계로 구성된다. 첫번째 단계에서, 그 레이어는 병렬으로 선형 activation의 집합을 생산하기 위해 몇몇의 컨볼루션을 수행한다. 두번째 단계에서, 각 선형 activation은 rectified linear activation function과 같은 비선형 activation function을 통하여 수행된다. 이 단계는 때떄로 detector 단계로 불린다. 세번째 단계에서, 우리는 pooling function을 아웃풋을 수정하기 위해 사용한다.

Pooling function은 현재 위치에서 아웃풋 근처의 요약 통계와 넽의 아웃풋을 대체한다. 예를들어, max pooling연산은 네모의 이웃들 내에 최대 아웃풋을 report한다. 다른 유명한 pooling functions은 직각의 이웃들의 평균을 포함한다. 네고 이웃의 L2 놈 또는 가중치된 평균은 중앙 픽셀로부터의 거리를 기반으로한다.

모든 경우에서, pooling은 재표현이 근사적으로 인풋의 작은 이동에 불변하도록 근사하는 것을 돕는다. 이동에 불변함은 만약 우리가 그 인풋을 작은 양 이동한다면 pool된 아웃풋의 대부분의 값이 변하지 않는 다는 것을 의미한다. 그림 9.8을 어떻게 이것이 작동하는지 봐라. 지ㅣ여역 이동에 불변함은 만약 우리가 어떤 특징이 그것이 있는 곳에서 정확히 더 존재하는지 아닌지에 대해 케어한다면 값에 유용할 수 있다. 예를들어, 이미지가 얼굴을 포함하는지 안포함하는지 결정할 때, 우리는 픽셀 단위의 정확한 눈의 위치를 알 필요가 없다. 우리는 단지 눈이 얼굴의 왼쪽에 있고 다른 눈은 오른쪽에 있다는 것을 알면 된다. 다른 문맥에서, 이것은 특징의 위치를 보존하는데 중요하다. 예를들어, 만약 우리가 특정한 방향에서 부합하는 두 엣로 정의된 코너를 찾길원한다면, 우리는 그 엣지들의 위치를 테스트하기 위해 충분히 잘 보존 할 필요가 있다.

풀링의 사용은 무한대로 강한 prior를 더함으로써 보여질 수 있다. 공간적인 영역하의 pooling은 이동에 불변함을 생상한다.

9.4 Convolution and Pooling as an Infinitely Strong Prior

Prior 확률 분포 컨셉을 다시 가져와보자

**Permutation invariant**.

Back prop 방법을 그대로 사용하는 CNN

9.7 Data Types

채널의 수와 차원에 따라 예측가능한 데이터가 다름.

9.9 Random or Unsupervised Feature P 363

V1? V2?

요즘의 많은 알고리즘은 특징 검출을 위해 Gabor filter를 많이 쓴다.