딥러닝의 구조와 설계

딥러닝의 설계와 학습 과정

목표 설정

딥러닝을 통해 성취할 목표를 설정해야 한다. 목표를 달성했는 지의 여부를 판단할 기준지표도 이 단계에서 같이 설정한다.

입력과 출력 정의

딥러닝에 주어질 입력과 출력의 종류와 형태를 정의한다. 딥러닝의 전체적인 구조와 성능에 큰 영향을 미치기 때문에, 사용가능한 데이터와 목표를 함께 고려하며 신중하게 선택할 필요가 있다.

데이터 준비

사용할 데이터를 준비한다. 앞서 정의한 입력 형태에 맞게 데이터를 가공할 필요가 있다. 너무 큰 이미지는 막대한 성능을 요구하므로 리사이징을 하거나, 컬러를 흑백으로 바꾸는 등 딥러닝에 더 적합하도록 데이터의 형식을 바꾸어야 할 때도 있다. 또한 언어 데이터의 경우 단어장을 만들어 각 단어가 하나의 숫자에 대응하도록 하거나 Word2Vec 등의 기존 솔루션을 활용한다.

학습 데이터와 검증 데이터의 분리

딥러닝의 특성상, 학습이 오랜 기간 진행되면 실제 딥러닝의 성능과 상관없이 학습에 사용된 데이터에 한해서는 높은 성취도를 보일 수 있다. 따라서 학습에 사용할 데이터와 검증에 사용할 데이터를 분리해야 만 정확하고 객관적인 검증이 가능하다. 편향을 막기 위해 전체 데이터를 임의로 추출해 검증용으로 쓸 수도 있으며, 목적에 따라 학습과 검증 데이터가 아예 다른 종류가 될 수도 있다.

딥러닝 레이어 설계

입력부터 출력 레이어에 이르기까지 딥러닝의 구조를 설계한다. 사용하는 데이터의 종류와, 목적에 따라서 적합한 구조가 달라지기 때문에 이들의 특성을 잘 알고 조합할 필요가 있다. 이론적으로 더 깊고 넓은 네트워크일수록 더 복잡한 일을 수행할 수 있으나, 반대급부로 학습에 더 오랜 시간이 걸리고, 더 많은 데이터와 더 빠른 하드웨어를 요구한다. 따라서 목적을 달성할 수 있는 네트워크의 적절한 규모가 어느 정도인지를 생각하며 설계해야 한다.

학습

딥러닝 라이브러리를 통해 설계한 네트워크를 학습시킨다. 학습 방법과 학습 횟수 등의 변수에 따라 학습 시간과 효율이 크게 달라지기 때문에 이 또한 데이터와 네트워크에 맞게 조정할 필요가 있다. 가능한 경우 GPGPU를 지원하는 하드웨어와 라이브러리를 사용해야 하며, AWS 등 Cloud 서비스를 이용할 수도 있다.

재설계

학습 결과를 토대로 딥러닝 네트워크를 평가하고, 부족한 부분을 보완할 수 있도록 재설계한다. 특히 딥러닝 네트워크의 성능은 대개의 경우 실제 학습을 거치기 전엔 예측하기 어려운 경우가 많기 때문에 많은 시행착오가 필요하다. 시간과 비용을 절약하기 위해 이러한 시행착오 중에는 전체 데이터 중 일부만을 사용하고, 의미 있는 성과가 나오면 전체 데이터를 이용한 학습을 진행할 수도 있다.

검증

학습 데이터로 학습한 결과를 검증 데이터를 통해 검증한다. 목표 설정 단계에서 정의한 기준 지표를 통해 딥러닝의 성취도를 평가한다. 또한 오류나 미흡한 특성을 보이는 데이터의 특징을 파악해 딥러닝의 한계를 파악한다.

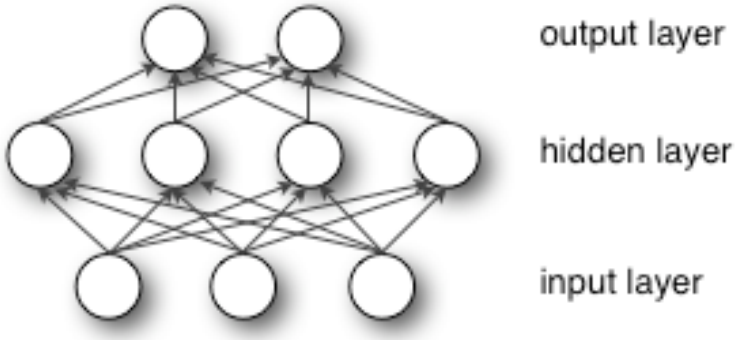
실사

학습한 딥러닝 네트워크를 통해 의미 있는 정보를 추출하거나, 별도의 서버나 하드웨어 등에 배치한다. 딥러닝은 학습은 높은 성능을 요구하지만 비해 사용에는 비교적 낮은 성능의 하드웨어로도 충분하므로, 서비스의 용도에 따라 다양한 하드웨어를 이용할 수 있다.

딥러닝 레이어

딥러닝의 구조는 딥러닝의 입력과 출력 사이에 어떤 레이어들이 쌓여 있는지를 의미한다. 각 레이어마다 다양한 특징을 가지고 있으며. 이를 잘 조합해야 데이터와 목적에 맞는 딥러닝을 설계할 수 있다.

일반적인 NN(Neural Network) 레이어 / Dense



바로 위의 네트워크의 노드(그림에서 동그라미)를 입력으로 받아 출력으로 지정된 개수만큼의 노드를 내는 레이어. 입력과 출력 노드 모두가 연결되어 있다. 이러한 NN을 많이 쌓아 DNN(Deep Neural Network)를 만들 수 있으며, 다른 레이어들과 조합하기도 용이하다.

가장 기본적인 레이어로, 분류, 변환, 회귀 등 이론적으로 충분한 수가 있으면 모든 함수를 따라할 수 있다. 그러나 계산의 효율성과 학습 측면에서의 한계도 비교적 명확하다. 따라서 다른 레이어들과 결합하여 사용되는 경우가 많다.

Activation

딥러닝 네트워크의 각 노드는 자기에게 들어오는 화살표, 즉 입력값을 각 입력에 대한 가중치를 곱한 뒤 모두 더해 계산한다. 이후 출력값을 어떻게 내보낼지를 정하는 것이 Activation 함수이다. 수치를 예측하기 위한 회귀분석이 목적이면 Linear를 쓰지만 대개의 경우 0근방에서 값이 변화가 큰 비선형적인 함수를 많이 쓰며, 최근에는 Linear에서 음수의 경우 0으로 만들어버리는 ReLU (Rectified Linear Unit)이 각광을 받고 있다.

Recurrent Neural Network (RNN)

일반적인 NN은 주어진 입력을 판단할 때 그 입력만을 고려해서 출력을 낸다. 다시 말해 맥락에 따른 판단이 불가능한 것이다. RNN은 이를 해결하기 위해 네트워크가 메모리를 가지고 이용하며, 입력한 내용이 한 번 쓰이고 끝나는 대신 다시 되새김질 되듯 네트워크에 재투입된다. 가령 1, 2, 3, 4, 3, 2, 1, 2, 3, … 으로 가는 수열을 예측할 때, 직전 숫자만 입력 받아 다음 숫자를 예측하는 것은 불가능하다. 2 다음에 3이 올 수도 있고 1이 올 수도 있기 때문이다. RNN은 2 이전의 4, 3를 같이 입력 받아 출력을 결정하기 때문에, 입력이 현재 하락하는 중이란 것을 인식할 수 있고, 정확한 예측을 할 수 있다.

따라서 이러한 맥락이 중요한 텍스트 분석이나 주가 분석, 필기체 인식 등에서 높은 성능을 보인다. 그러나 같은 크기의 다른 네트워크보다 더 많은 매개변수를 가지고 있어 학습에 시간이 오래 걸리고 난이도가 높다.

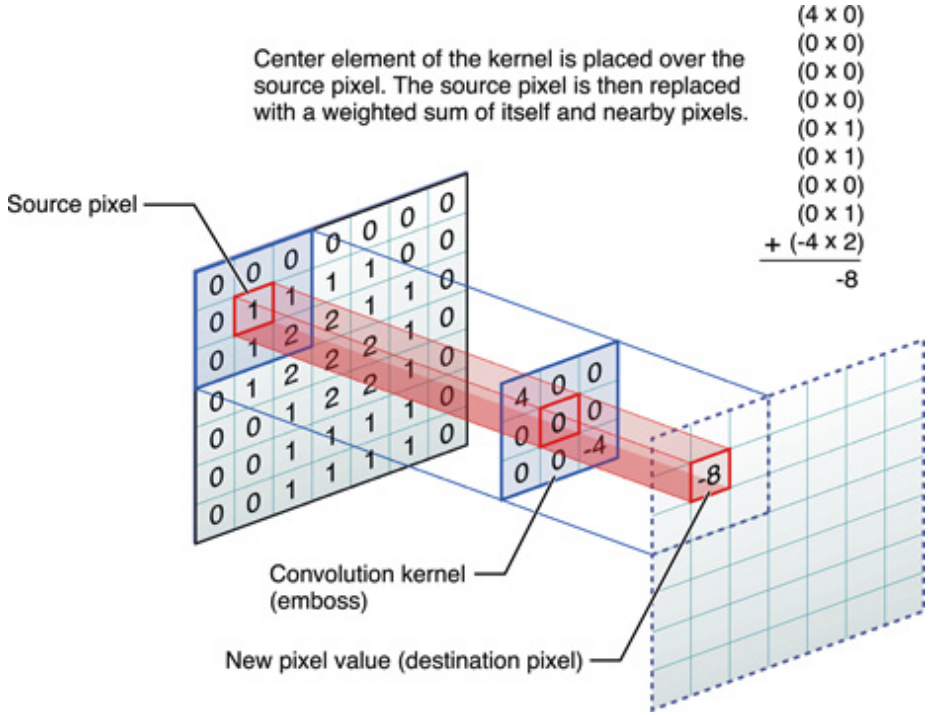
LSTM (Long Short Term Memory)

모든 딥러닝은 네트워크의 출력과 정답(목표값)을 비교하여, 정답에 가까워질 수 있는 방향으로 각 레이어의 계수를 조정하는 것을 통해 학습을 한다. 이 때 이 방향은 미분을 통해 기울기로 계산되며, 이 기울기에 일정한 학습률 (Learning Rate)를 곱하고 입력의 크기만큼 가중치를 넣어 조금씩 계수를 변화시킨다.

문제는 이 변화의 방향 혹은 가야할 방향의 경사 (Gradient)를 계산할 때, RNN의 경우 네트워크가 여러 입력에 걸쳐 출력을 계산하기 때문에 출력에서 먼 부분의 입력에 대해서는 Gradient가 점점 감소하여 0이 되거나, 혹은 점점 커지면서 무한대로 발산하는 문제가 발생한다. 쉽게 말하면 글의 내용을 인식할 때, 글 앞부분의 내용에 대해서는 학습의 방향이 제대로 전달되지 않는 것이다.

이를 극복하기 위해 뉴런 내부에 이전 뉴런의 상태를 전달할 지 말지 정하는 관문을 설치하고 이 관문의 작동도 학습시키는 LSTM이 등장하였다. 사람의 기억을 딥러닝에 구현한 것이다. 특히 LSTM은 문장의 구조적인 요소를 학습시키는데 효과가 크다. (괄호를 열고 적절하게 다시 닫아주는 등 기존의 RNN으로는 힘들었던 것을 잘 할 수 있다)

Convolution Neural Network (CNN)



컨볼루션 레이어는 특히 이미지를 인식하는 것에 큰 강점을 가지며, 사진 및 영상 데이터를 처리할 때 필수적으로 사용된다. 컨볼루션 레이어는 이미지의 각 좌표마다, 특정한 크기만큼 점과 주변 점들을 뽑아내고, 그 값에 필터를 적용한 값을 다시 내보내는 것을 통해 이미지의 다양한 특징을 추출한다.

가령 수직선을 강조하기 위해서는 다음과 같은 필터를 사용한다.

−1	0	1
−1	0	1
−1	0	1

만약 주변 픽셀이 다 같은 색이면, 한 쪽은 1, 한 쪽은 −1이게 상쇄되어 0이 나타난다. 그러나 왼쪽은 검은색(값이 0), 오른쪽은 흰색(값이 1)이라고 하자. 그러면 컨볼루션한 값은 (−1) * 0 + (−1) * 0 + (−1) * 0 + 1 * 1 + 1 * 1 + 1 * 1 = 3이며, 반대로 왼쪽이 흰색, 오른쪽이 검은색일 경우 (−1) * 1 + … + 1 * 0 + … = −3으로 수직경계선이 아닌 곳에서는 0에 가까운 값이, 수직경계선에서는 절대값이 큰 값이 나타나 구분이 가능하다.

이미지 분류에서의 사례

입력

입력받을 데이터의 크기를 정해야 한다. 가령 (128, 128, 3)의 크기라면 가로 128, 세로 128 해상도의 RGB 이미지를 입력으로 받는 것이 된다. 만약 준비된 데이터가 크기가 다른 경우, 리사이징하거나 비율이 안 맞는 부분을 흰색이나 노이즈로 채울 필요가 있다.

컨볼루션

필요한 만큼 컨볼루션 레이어를 쌓는다. 컨볼루션 레이어에서 중요한 것은 필터의 크기다. 필터가 클수록 한 점의 값을 더 많은 점을 참고해 계산할 수 있어 더 많은 정보를 참고할 수 있으나, 그만큼 더 긴 계산 시간이 필요하고, 학습에도 오랜 시간이 걸린다.

이를 해결하기 위해 Pooling 레이어를 추가할 수도 있다. Pooling 레이어는 이미지를 특정한 크기의 칸으로 나누고, 칸 안에서 Max값을 뽑아내거나 Mean값을 뽑아낸다. 만약 4 * 4 이미지에 크기가 2*2인 Pooling을 하면 총 4개 칸으로 나눠지고, 원래 4 *4 = 16에서 4개로 다루는 숫자가 줄어들면서 원래 이미지의 중요한 특징을 유지할 수 있다.

출력

컨볼루션과 풀링 이후, 최종적으로 결과를 내기 위해 NN를 마지막에 추가한다. 이 때, NN의 출력 개수는 분류의 가지수와 같아야 한다. 즉 개와 고양이, 금붕어를 구분하는 딥러닝이라면 마지막 출력의 개수가 3이어야 하는 것이다. 가령 출력이 (0.2, 0.3, 0.5)로 나왔을 경우 가장 높은 값인 0.5가 이 딥러닝의 예측이며, 이것은 금붕어라는 것을 의미한다.

학습 및 검증

전체 이미지 셋에서 학습용 데이터와 검증용 데이터를 분리한 뒤, 충분한 시간을 들여 학습한 뒤 정확도를 평가한다.