인공지능개론 정리노트#5

201904022 김상옥

202284026 안정빈

질문1. 합성곱 신경망(CNN)과 순환 신경망(RNN)의 차이점과 각각의 주요 적용 사례는 무엇인가요?(7장)

김상옥: 수업시간에는 합성곱을 이용하여 CNN을 배웠어. 전에 질문/답변을 했을 때는 CNN이 이미지 처리에 매우 강점을 가진 신경망이라고 결론을 내기도 했었는데 요즘 사용하는 AI들은 RNN을 사용한다고 많이 들었어. 그럼 RNN이 어떤 신경망이고, CNN과의 차이가 무엇일까?

안정빈: RNN은 순환 신경망이야. 순차 데이터 입력을 처리하고 특정 순차 데이터 출력으로 변환하도록 훈련된 딥러닝 모델이라고 해. RNN은 시퀀스 데이터나 순차데이터(텍스트, 음성 데이터 등)을 처리하는데 적합하고 CNN은 이미지 데이터나 공간적 데이터(영상, 2D/3D 이미지 등)를 처리하는 데 적합해. RNN은 순환 구조를 가지고 있고, 시간 축을 따라 이전 상태를 현재 상태에 전달하CNN은 합성 곱 층과 풀링 층을 사용하여 공간적 특징을 추출하고 계층적으로 학습하는 구조를 가지고 있어. RNN은 시간적 순서가 중요한 문제에서 주로 사용되고 CNN은 공간적 구조에 사용되는 것 같은데 각각 주요 적용 사례는 어떻게 되는 것 같아?

김상옥: CNN은 영상처리에 강점이 있는 만큼 영상처리에 많이 사용될 거 같아. 우선 영상처리라는 것이 매우 활용범위가 넓은 기술로 사람이 눈으로 보고 판단 후 행동하듯이 기계의 눈이 되는 것이 이미지(영상)이고, 영상처리는 그 이미지에서 사물을 인식하든 특정한 무언가를 인식하고 판단하게 되는 구조야. 그렇기 때문에 무언가를 보고 판단해야 하는 문제에 있어서는 CNN이 중요하다고 생각하고 대표적으로 자율 주행이 CNN의 적용 사례라고 생각해. RNN을 순차적으로 데이터를 받아들이며 텍스트, 음성에 강하다고 답변해줬는데 지금의 Chat GPT도 그렇고 최근에 발표된 GPT 4o도 음성을 통한 커뮤니케이션이 매우 발전한 것으로 볼 때 RNN은 사람과의 소통에 있어 중요한 거 같아. 결국 CNN과 RNN모두 하나만 사용하는 것이 아니라 잘 조합해서 사용해야 하지 않을까? 그런 점에서 너의 생각은 어때?

안정빈: 나도 CNN과 RNN을 조합해서 사용하면 좋은 시너지를 얻게 될 것이라고 생각해. 실제로 조합해서 다양한 분야에서 활용하고 있는 것 같아. 비디오 데이터를 분석하는 경우에는 비디오의 각 프레임에서 CNN이 공간적 특징을 추출하고, RNN이 이러한 특징들의 시퀀스를 분석하여 특정 동작을 인식해. 질병 진단 및 설명을 하는 경우에는 CNN을 사용해서 의료 영상(CT, MRI)에서 병변을 탐지하고, RNN을 사용하여 탐지된 병변에 대한 설명을 생성하여 활용한다고 해. 이렇게 조합해서 활용하면 복잡한 문제를 효과적으로 해결할 수 있는 것 같아. 각 특성인 CNN은 공간적 특징을 잘 추출하는 것과, RNN이 시간적 또는 순차적 데이터를 잘 처리하기 때문에, 이들의 결합은 특히 시각적 데이터와 시간적 데이터를 동시에 처리해야 하는 문제에서 유용할 것 같아.

김상옥: 나도 그 의견에 동의해. 결국 장점들을 잘 조합해서 AI를 만드는 것이 중요할 거 같고, 나중에는 또 이 둘을 합친 새로운 신경망이 나타날 지도 모르는 일이야. 분명한 점은 현재에서 앞

으로도 이런 신경망의 구조가 지속적으로 발전할 것이라는 점이고, 그런 신경망이 적용될 분야가 무궁무진하다는 점이야. 그렇기 때문에 지금 배우는 신경망의 기초를 잘 잡고 간다면 나중에 개발자로써 AI를 활용하거나 이용하여 무언가를 만든다고 할 때 분명 도움이 될 거라고 생각해. 그런 점에서 AI의 눈이 되어주는 CNN, 소통을 위한 RNN의 발전은 일상과 일에서 지금보다 더 많이 사용하게 될 거라고 봐.

결론: 두 모델의 차이점은 CNN은 공간적 특징을 잘 추출하고 RNN은 시간적 또는 순차적 데이터를 잘 처리합니다. CNN은 영상처리, RNN은 텍스트, 음성 처리 등에서 많이 사용합니다. 두가지의 장점들을 잘 조합해서 시각적 데이터, 시간적 데이터를 동시에 처리하는 문제에서 유용하게 활용할 수 있습니다. 이러한 조합으로 다양한 발전이 생길 것 같습니다.

질문2. 딥러닝 모델이 실제 세계의 문제를 해결할 때 자주 겪는 어려움들은 무엇인가요?(8장)

안정빈: 딥러닝 모델은 여러 층으로 데이터를 학습하고 예측하는 신경망이야. 우리가 실제로 어떤 문제에서 딥러닝 모델을 활용할까?

김상옥: 네트워크. 하드웨어의 발전 등 딥러닝을 위한 기술도 같이 발전하면서 현재는 엄청난 성장을 이루었고, 현재에도 GPT를 활용하여 개인에게 있어 업무, 과제에 활용하기도 해. 이외에도 물류, 공장, 의류 등.. 사실상 AI라는 것이 결국 사람같은 지능을 만들고, 사람의 일을 대체하기 위해 발전해왔기 때문에 딥러닝은 사실상 사람의 모든 일에 활용될 수 있다고 생각해.

안정빈: 맞다고 생각해. 나는 실생활에서 ChatGpt와 siri,를 자주 사용하는 편이야. 내가 사용했을 때는 ChatGpt는 내가 질문한 의도와 다른 답변을 하는경우가 있고, siri는 내가 부르는 것을 못들을 때와 질문을 이해를 못하는 경우가 있어. 이렇게 이용하면서 불편할 때가 가끔 생기는데, 딥러닝 모델을 사용해서 실생활에서 문제를 해결하려고 할때 겪어본 어려움이 있어?

김상옥: 나도 같은 경우로 Chat GPT를 사용할 때 의도하지 않은 답변이나 잘못된 정보를 얻은 적이 있어. 그럴 때는 교차검증을 통해서 정보를 다시 확인해야 하는 불편함이 있었어. 물론 아직 완벽하지 않고 발전하는 단계라는 것을 이해하고 있어서 그런 불편함은 이해하고 사용하고 있어. 그런데 최근에 GPT 4o가 나오면서 무료 사용자도 일정 시간마다 GPT 4이상의 성능을 경험할 수 있었는데 확실이 GPT 3.5와 4, 4o의 차이가 매우 크다고 느껴졌어. 정보의 질도 그렇고 전체적으로 답변이 깔끔하고 명확하다는 느낌을 받았는데 이런 정보의 정확도나 답변의 문법, 단어 등 딥러닝은 앞으로 계속해서 인간 친화적인 상호작용을 추구할 거라고 생각하고 그럴수록 불편함, 어려움은 점점 줄어들 거라고 생각해. 이런 상호작용의 어려움 말고 다른 어려움은 어떤 것이 있을까?

안정빈: 딥러닝은 데이터를 학습하는 것이라서 나도 사용을 하면서도 데이터를 다룰 때 프라이버

시와 관련된 윤리적 문제가 유발할 것 같다는 생각이 들어. 이러한 문제에 대해서도 규제가 필요하다고 생각해. 그리고 딥러닝을 학습할때 데이터를 돌릴 경우에는 컴퓨터의 성능도 중요하다는 것을 알게됐어. 딥러닝 모델 훈련을 할 때 GPU나 TPU가 필요한데 이런 하드웨어의 비용이 많이 들고, 복잡한 모델은 훈련시간도 매우 오래걸려. 이렇게 다양한 어려움이 아직 있는 것 같아.

김상옥: 현실적으로 지금은 하드웨어에서도 물리적인 자원이 엄청난 비용이 들어간다는 것 본 적이 있어. 특히 GPU, NPU의 가격이나 그것을 이용하여 결국에는 전기를 사용하는데 AI학습을 위한 전기 사용량이 상상도 못할 정도로 많이 나온다고 들었어. 그리고 딥러닝을 사용하는데 있어서 개인, 윤리적인 문제도 물론이고 그렇기 때문에 해킹 공격에 대한 보안의 중요성도 매우 중요할 것 되겠지. 결국 딥러닝의 가능성은 무한하지만 아직까지는 어느정도 한계에서 활용이 된다는 점이 지금의 어려움일 것이고, 그런 어려움을 해결하는 것이 앞으로의 문제라고 생각해.

결론: 현재 딥러닝은 학습을 위해 강력한 하드웨어와 그로 인해 발생하는 전기, 냉각비용 등 엄청난 자원을 소모하고 있다. 현실적으로 아직까지는 대규모의 신경망을 학습하기 위해 물리적인 어려움이 있으며, 현재 많이 사용하는 GPT같은 딥러닝 모델들의 경우 사용자에게 완벽한 상호작용을 제공하지 못하며, 정보의 정확도 또한 아직까지는 완전한 신뢰를 주지는 못한다. 앞으로의 딥러닝에서도 개인의 프라이버시, 윤리적인 문제 그리고 해킹의 공격을 막는 보안 문제 또한 해결해야 할 어려움이 많이 있을 것으로 생각합니다.

질문3. 인공지능의 발전을 위한 다른 최신 기술들은 무엇이 있을까?

안정빈: 인공지능 분야에서 다양한 모델들이 지속적으로 발전하고 있어. 예를들어 GPT모델은 OpenAI가 개발한 GPT시리즈는 우리가 많이 사용하고있지. 다양한 자연어 이해 및 생성 작업에서 뛰어난 성능을 보여주고 있어. 다른 최신 기술들은 무엇이 있을까?

김상옥: 프로그래밍 코드를 작성하는 AI, 이미지 생성, 음성 인식, 최근에는 영상을 만드는 AI까지 각종 AI를 활용한 기술들이 매우 많이 나타나고 있어. 확실히 AI의 잠재력이 뛰어나지 때문에 앞으로 사람이 하는 일을 대체하는 것에는 시간 문제라고 생각해. 이렇게 인공지능을 활용하는 분야가 많은데 그렇다면 인공지능의 발전을 도울 수 있는 다른 분야의 기술은 무엇이 있을지 궁금해. 지금 생각나는 것은 더 빠른 연산을 위한 하드웨어의 개발이 지금에서는 주요한 기술로 보여지고 있는데 다른 기술은 무엇이 있을까?

안정빈: 강화학습 기술도 인공지능의 발전을 도울 것 같아. 강화학습 기술은 기계 학습의 한 영역인데, 에이전트 스스로 환경에서 주는 보답을 학습해 나아가서 최적화하는 기법이야. 이 강화학습 기술은 인공지능이 환경과 상호작용하며 학습하는데 중요한 역할을 해. 우리가 앞으로 인공지능과 살아가면서 다양한 문제점들이 생기는데 이러한 기술이 발전하면 인공지능을 모든 분야에서도 활용하는 날이 올 것 같아. 그리고 교수님께서 AI 모델로 산업 구조가 바뀔 것이라고 하셨어. 예

를들어 ON 디바이스, AI 로봇, 밀리터리 등을 꾸준히 개발하고 있어. 이러한 발전은 우리에게 어떻게 다가올 것 같아?

김상옥: 나도 생각해본 기술로는 물리적인 한계를 극복하기 위해 작년에 이슈가 된 초전도체, 혹은 양자 컴퓨터 같은 아직은 비현실적인 이야기지만 이런 기술이 발전한다면 AI에 있어 엄청난 성장을 가져올 거야. 물론 이런 기술은 AI뿐만 아니라 인류의 성장 속도를 매우 앞당길 기술이기도 해. 답변해준 것 처럼 On디바이스나 밀리터리에서도 AI가 접목되면 좋을 기술들이 매우 많을 거야. 그런 발전은 앞으로의 인간의 삶의 패턴을 많이 바뀌게 하지 않을까? 지금의 식당에서 사용하는 키오스크나 스마트폰이 없던 시절에서 스마트폰이 당연하게 된 시절처럼 AI도 언젠가는 당연한 기술이 될 것이야. 그런 점에서 각종 하드웨어 기술의 발전이 중요하다고 생각이 들어. 지금 일상에서 AI는 소프트웨어에서 주가 되지만, 앞으로 AI는 하드웨어가 주가 되는 새로운 형태의 AI가 나타나지 않을까?

안정빈: 답변처럼 AI도 당연한 기술이 되고, 앞으로 하드웨어가 주가 되는 새로운 형태의 AI가 나타날 수도 있다고 생각해. 특화된 하드웨어 GPU, TPU가 AI 작업을 더 효율적으로 수행할 수 있어. 그리고 신경망 프로세서 및 가속기의 개발이 확대되고 있는데 이러한 하드웨어는 딥러닝 모델의 학습 및 추론 작업을 빠르고 효율적으로 수행할 수 있도록 설계되고 있어. 하드웨어의 발전이 더욱 빠르게 진행되고 있는 것 같아. 이렇게 발전하다 보면 하드웨어가 AI 발전의 중심이 되고 새로운 다양한 형태의 AI도 나올 것 같아.

결론: 인공지능의 발전을 위해 하드웨어의 기술이 매우 중요하다고 생각합니다. 우선 GPU, NPU의 발전이 곧 인공지능의 발전으로 이어진다고 생각합니다. 강력한 하드웨어는 인공지능의 학습과 시간이 효율적으로 나타나게 되기 때문입니다. 다른 하드웨어 분야의 발전으로 밀리터리 기술, 새로운 AI로봇, 휴머노이드 등 각종 하드웨어 분야에서도 발전을 통해 AI와 접목된다면 인공지능의 발전에도 도움이 될 거라고 생각합니다. 현실적이지는 않지만 더욱 미래를 본다면 양자 컴퓨터, 작년에 이슈였던 초전도체같은 기술은 AI의 성장에 엄청난 박차를 가할 것으로 생각되는 기술입니다. 그 또한 물리적인 한계를 극복하기 위한 기술로 앞으로 인공지능의 발전에는 각종 NPU, 하드웨어의 발전도 매우 중요할 것으로 생각합니다

질문4. 가중치를 -1, 0, 1로 제한하여 사용하는 1.58bit AI가 정말 효율적일까?

김상옥: 최근에 본 뉴스인데 AI학습을 할 때 가중치를 실수가 아닌 정수 (-1, 0, 1)만 사용하여 학습을 했을 때 실수 가중치에 비해서 크게 차이가 나지 않는 성능을 낸다고 들었어. 이에 대해 자세하게 알지는 못하지만 우리가 지금까지 배운 학습을 바탕으로 가중치를 -1, 0, 1로만 제한한다면 신경망의 구조와 그 효율성이 기존 신경망과 어떻게 다를 지 궁금해.

안정빈: 가중치의 범위가 제한되면 역전파 알고리즘을 활용해서 가중치를 효율적으로 업데이트

하는 것이 어려워질 것 같아. 기존 신경망은 가중치의 값을 임의의 실수로 설정할 수 있어. 이는 모델이 다양한 함수를 학습하고 데이터에 대한 복잡한 패턴을 표현하는데 도움이 되는데 가중치를 -1,0,1로 제한하면 모델은 학습이 감소되고 표현도 감소될 것 같아.

김상옥: 확실히 지금으로서는 -1, 0, 1로만 학습을 한다고 하면 실수보다 표현 범위가 매우 한정적이여서 MNIST 손글씨 학습도 제대로 이루어지지 않을 거라고 생각이 들었어. 혹은 그런 한계를 극복하기 위해 실수 신경망보다 훨씬 넓고 깊은 신경망의 구조로 이루어져야 하지 않을까? 하지만 연구 결과에서 나름 유의미한 결과를 제시한 것으로 보면 우리가 모르는 다른 방법이 있을 거라고 생각해. 그런 점에서 만일 (-1, 0, 1)로 제한한 AI가 지금의 AI와 크게 성능차이가 없다고 가정한다면 기존보다 어떤 점이 효율적일까?

안정빈: 가중치를 -1,0,1로 제한하면 가중치의 표현에 필요한 비트 수가 줄어들어 메모리 사용량을 줄일 수 있어. 이는 모델의 크기를 줄이고 모델을 배포 및 실행할 때 더 적은 메모리를 필요로 해. 그리고 연산 량이 줄어들어서 연산 속도가 향상되고 모델의 설계가 단순화되어 구현 및 유지 보수를 간편하게 할 수 있어. 여러면에서 계산비용이 급감시킬 수 있는 것같아. 이러한 효율적인 면들은 리소스 제한이 있는 환경에서 유용할 것 같아.

김상옥: 나도 그 의견에 동의해. 기존의 실수로 연산하던 부분을 -1, 0, 1로만 제한한다면 연산에 사용되는 시간, 자원의 비용이 매우 효율적으로 바뀔 거라고 생각해. 그건 대규모 신경망에서는 말도 안 될 정도로 차이가 나지 않을까? 기존의 신경망이 100의 비용을 소모한다면 -1, 0, 1의 연산으로 학습한다면 5의 비용을 소모하는 정도의 차이가 날 것 같아. 여기서 비용은 연산의 시간도 시간이지만 연산 량이 줄어드니 하드웨어의 비용도 줄어들고 전기 비용이나 냉각 비용을 포함해서 전체적 학습의 비용을 말해. 결과적으로는 이 -1, 0, 1로 학습이 기존 신경망과 비슷한 성능을 낼 수 있다라면 무조건적으로 -1, 0, 1로 학습을 해야 하겠지만, 아무래도 연구의 초기 단계이고 그 실현 가능성이 높지 않기 때문에 아직까지는 많이 알려지지 않은 것 같아.

안정빈: 올해 초에 나온 모델이 더라고. 나도 이번에 토론을 하면서 처음 알게 되었어. 성능, 필요한 메모리 양, 대기 시간, 행렬 연산 비용의 시간 모두 줄이는 데 성공했 어. 논문에서 '1비트의 대규모 언어 모델용 새로운 하드웨어 설계로의 문을 열었다.'라고 말했는데 정말 앞으로 기술 변화가 놀라울 것 같아. 1.58bit AI는 다른 모델과 동일한 성능을 내는데 필요한 비용의 일반 대규모 언어 모델에 비해 급감한다는 것은 메리트가 있는 것 같아.

결론: 1.58bit AI학습이 기존 신경망과 차이가 크지 않다면 무조건적으로 효율적인 학습입니다. 왜냐하면 실수 연산이 아닌 가중치를 -1, 0, 1로 제한하여 사용하기 때문에 기존보다 매우 적은 비용으로 같은 결과를 낸다는 것을 의미하기 때문입니다. 현재 인공지능의 문제인 물리적인 문제를 극복할 수 있는 강력한 학습방법이기 때문에 가능하다는 전재일 경우 매우 효율적인 학습 방법입니다. 하지만 아직까지는 연구가 많이 진행되지 않았고, 그 실현 가능성이 높지 않다고 생각을 하기 때문에 당장으로써는 사용할 이유는 없겠지만 앞으로 관심을 가질만한 주제라고 생각합니다.

안정빈

4차원 배열, 4차원 데이터

[5]:
```
import numpy as np

#CNN에서 계층 사이를 흐르는 데이터는 4차원임
#데이터 형상이 (10,1,28,28)이라면, 높이 28, 너비 28, 채널 1개인 데이터가 10개임
x=np.random.rand(10,1,28,28)
x.shape
```

[5]:
```
(10, 1, 28, 28)
```

[6]:
```
#10개의 데이터 중에 첫 번째 데이터에 접근하려면 단순히 x[0]이라 씀
x[0].shape
```

[6]:
```
(1, 28, 28)
```

[7]:
```
#첫 번째 데이터의 첫 채널의 공간 데이터에 접근시
x[0,0]
```

[7]:
```
array([[0.49521615, 0.61800434, 0.75760753, 0.82068517, 0.56819523,
        0.05908713, 0.08066626, 0.33005808, 0.97673157, 0.20106956,
        0.07464219, 0.57532127, 0.17682773, 0.29513768, 0.47701519,
        0.59566278, 0.60108215, 0.26711222, 0.63551062, 0.2167237 ,
        0.68870407, 0.08667564, 0.00230929, 0.23328987, 0.71831438,
        0.36272552, 0.41574046, 0.44595366],
       [0.94575766, 0.62909462, 0.05013236, 0.24624883, 0.52789548,
        0.7655844 , 0.97917882, 0.62675222, 0.32308081, 0.24451424,
        0.81915304, 0.62606568, 0.78854141, 0.37320229, 0.48934216,
        0.18326948, 0.8086537 , 0.68360815, 0.70106686, 0.63481117,
        0.70220599, 0.42043123, 0.39150773, 0.60326239, 0.92046312,
```

합성곱 연산 구현의 문제점

- for문을 겹겹이 써야함
- 넘파이에 for문을 사용하면 성능이 떨어진다는 단점도 있음

im2col 함수를 사용한 구현

- for 문 대신에 사용함
- 입력 데이터를 필터링(가중치 계산)하기 좋게 전개하는(펼치는) 함수임
- 3차원 입력 데이터에 im2col을 적용하면 2차원 행렬로 바뀜

im2col 입력 전개

- im2col은 필터링하기 좋게 입력 데이터를 전개함
- 입력 데이터에서 필터를 적용하는 영역(3차원 블록)을 한 줄로 늘어 놓음
- 실제 상황에서 영역이 겹치는 경우가 대부분임
- im2col을 사용해 구현하면 메모리를 더 많이 소비하는 단점이 있음
- 문제를 행렬 계산으로 만들면 선형 대수 라이브러리를 활용해 효율을 높일 수 있음
- im2col 로 입력 데이터를 전개한 다음에는 합성곱 계층의 필터를 1열로 전개하고, 두 행렬의 곱을 계산 하면 됨

im2col 함수의 인터페이스
im2col(input_data, filter_h, filter_w, stride =1, pad = 0)

- input_data - (데이터 수, 채널 수, 높이, 너비)의 4차원 배열로 이루어진 입력 데이터
- filter_h - 필터의 높이
- filter_w - 필터의 너비
- stride - 스트라이드
- pad - 패딩

im2col 실제 사용

- 첫 번째는 배치 크기가 1(데이터 1개), 채널은 3개, 높이 너비가 7x7 의 데이터
- 두 번째는 배치 크기가 10이고 나머지는 첫 번째와 같음
- im2col 함수를 적용한 두 경우 모두 2번째 차원의 원소는 75개임(채널3개, 5x5 데이터)

```
import sys, os
import numpy as np
sys.path.append(os.pardir)
```

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).

    Parameters
    ----------
    input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비)
    filter_h : 필터의 높이
    filter_w : 필터의 너비
    stride : 스트라이드
    pad : 패딩

    Returns
    -------
    col : 2차원 배열
    """
    N, C, H, W = input_data.shape
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1

    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride*out_h
        for x in range(filter_w):
            x_max = x + stride*out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
    return col
```

```
#배치 크기가 1(데이터1), 채널 3개, 높이와너비 7x7의 데이터
x1 = np.random.rand(1,3,7,7)
col1 = im2col(x1,5,5,stride=1, pad = 0)
print(col1.shape)

# 배치크기만 10이고 나머지는 첫번째와 동일
x2 = np.random.rand(10,3,7,7)
col2 = im2col(x2,5,5,stride=1, pad = 0)
print(col2.shape)



#im2col 함수를 적용한 두 경우 모두 2번째 차원의 원소는 75개임(채널3개, 5x5 데이터)
```

```
(9, 75)
(90, 75)
```

im2col 사용하여 합성곱 계층 구현 필터는 (FN,C,FH,FW)의 4차원 형상

(1) 입력 데이터를 im2col로 전개
(2) 필터도 reshape을 사용해 2차원 배열로 전개
(3) 전개한 두 행렬의 곱을 구함
(4) 출력 데이터를 적절한 형상으로 바꿔줌(transpose 함수를 사용하여 다차원 배열의 축 순서를 바꿈)

```
class Convolution:
    def __init__(self,W,b,stride = 1, pad = 0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

    def forward(self,x):
        FN,C,FH,FW = self.W.shape
        N,C,H,W = x.shape
        out_h =  1+int((H+ 2*self.pad - FH) / self.stride)
        out_w =  1+int((W+ 2*self.pad - FW) / self.stride)

        col = im2col(x,FH,FW,self,stride , self, pad)  #(1)
        col_W = self.W.reshape(FN, -1).T #(2)
        out = np.dot(col,col_W) + self.b #(3)

        out = out.reshape(N, out_h, out0_W, -1).TRANSPOSE(0,3,1,2) #(4)

        return out
```

```
class Pooling:
    def __init__(self, pool_h, pool_w, pada=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

    def forward(self,x):
        N,C,H,W = X.shape
        out_h = int(1+(H - self.pool_h)/ self.stride)
        out_w = int(1+(W - self.pool_w)/ self.stride)

        #(1) 전개
        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1,self.pool_h*self.pool_w)

        #(2) 최댓값
        out = np.max(col, axis =1)
        #(3) 성형
        out = out.reshape(N, out_h, out_w, C).transpose(0,3,1,2)

        return out
```

풀링 계층 구현

- 합성곱 계층과 마찬가지로 im2col을 사용해 입력 데이터를 전개함
- 풀링의 경우엔 채널 쪽이 독립적으로 전개한다는 점이 합성곱 계층 때와 다름

풀링 계층 구현의 흐름

- 전개한 행렬에서 행결 최대값을 구하고 적절한 형상으로 성형하면 됨

풀링 계층 구현 세 단계
(1) 입력 데이터 전개
(2) 행렬 최댓값 구함
(3) 적절한 모양으로 성형

SimpleConvNet의 초기화

- conv_param -합성곱 계층의 하이퍼파라미터는 딕셔너리 형태로 주어짐
- weight_init_std - 초기화 때의 가중치 표준편차

```
class SimpleConvNet:
    def __init__(self, input_dim=(1,28,28),
                 conv_param={'filter_num':30,'filter_size':5,
                             'pad':0, 'stride':1},
                 hidden_size=100, output_size=10, weight_init_std=0.01):
        filter_num = conv_param['filter_num']
        filter_size = conv_param['filter_size']
        filter_pad = conv_param['pad']
        filter_stride = conv_param['stride']
        input_size = input_dim[1]
        conv_output_size = (input_size - filter_size + 2*filter_pad) /filter_stride+1
        pool_output_size = int(filter_num*(conv_output_size/2)*(conv_output_size/2))

        #가중치 초기화
        # 학습에 필요한 매개변수는 1번째 층의 합성곱 계층과 나머지 두 연결 계층의 가중치와 편향,
        #이 매개변수 들을 인스턴스 변수 params 딕셔너리에 저장함
        # 1,2,3층의 가중치와 편향을 키로 각각 저장함

        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
        self.params['b1'] = np.zeros(filter_num)
        self.params['W2'] = weight_init_std * np.random.randn(pool_output_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b3'] = np.zeros(output_size)

        #CNN을 구성하는 계층들을 생성
        # 순서가 있는 딕셔너리인  layers에 계층들을 차례로 추가함
        # 마지막 SoftmaxWithLoss 계층만큼은 last_layer라는 별도 변수에 저장함
        self.layers = OrderedDict()
        self.layers['Conv1'] = Convolution(self.params['W1'].self.parmas['b1'],
                                           conv_param['stride'].conv_param['pad'])
        self.layers['Relu1']=Relu()
        self.layers['Pool1']= Pooling(pool_h=2, pool_w=2, stride=2)
        self.layers['Affine'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2']= Relu()
        self.layers['Affine2'] = Affine(self.parmas['W3'].self.params['b3'])

        self.last_layer = SoftmaxWithLoss()
```

추론을 수행하는 predict 메서드

- 초기화 때 layers에 추가한 계층을 맨 앞에서부터 차례로 forward 메서드를 호출
- 그 결과를 다음 계층에 전달

```
def predict(self,x):
    for layer in self.layers.values():
        x=layer.forward(x)
    return x
```

손실 함수를 구하는 loss 메서드

- 인수 x는 입력 데이터, t는 정답 레이블
- predict 메서드의 결과를 인수로 마지막 층의 forward 메서드를 호출
- 첫 계층부터 마지막 계층까지 forward를 처리

오차역전파법으로 기울기 구현

- 매개변수의 기울기는 오차역전파법으로 구함
- 순전파와 역전파를 반복
- grads라는 딕셔너리 변수에 각 가중치 매개변수의 기울기를 저장
- MNIST 데이터 셋으로 학습시 훈련데이터 정확도 99.8%, 시험 데이터 정확도 98.96%
- CNN이 이미지의 공간적인 형상에 담긴 특징을 잘 파악함을 볼 수 있음

```
def gradient(self,x,t):
    #forward
    self.loss(x,t)

    #backward
    dout = 1
    dout = self.last_later.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    #결과 저장
    grads={}
    grads['W1'] = self.layers['Conv1'].dW
    grads['b1'] = self.layers['Conv1'].db
    grads['W2'] = self.layers['Affine1'].dW
    grads['b2'] = self.layers['Affine1'].db
    grads['W3'] = self.layers['Affine2'].dW
    grads['b3'] = self.layers['Affine2'].db

    return grads
```

학습전과 후의 가중치를 비교

- MNIST 데이터셋으로 간단히 CNN 학습을 수행시 1번째 층의 가중치의 현상이 (30,1,5,5)임(필터 30개, 채널 1개, 5X5 크기)
- 채널이 1이라는 것은 이 필터를 1채널의 회색조 이미지로 시각화 할 수 있다는 뜻
- 합성곱 계층 필터를 이미지로 나타내 봄
- 학습 전 필터는 무작위로 초기화되고 있어 흑백의 정도에 규칙성이 없음
- 학습을 마친 필터는 규칙성 있는 이미지가 되었음
- 학습 후 규칙성 있는 필터로 변경됨

규칙성 있는 필터는 무엇을 보고 있는 것일까?

- 에지(색상이 바뀌는 경계선)와 블롭(국소적으로 덩어리진 영역)을 보고 있음
- 필터1은 세로 에지에 반응, 필터2는 가로 에지에 반응하는 것을 알 수 있음
- 합성곱 계층의 필터는 에지나 블롭 등의 원시적인 정보를 추출 할 수 있음
- 원시적인 정보가 뒷단 계층에 전달되는 것이 CNN에서 일어나는 일

CNN의 각 계층에서 어떤 정보가 추출되는가?

- 계층이 깊어질수록 추출되는 정보(정확히는 강하게 반응하는 뉴런)는 더 추상화 된다는 것을 알 수 있음
- 1번째 층의 합성곱 계층에서는 에지나 블롭 등의 저수준 정보가 추출됨

AlexNet의 각층의 추출 정보

- 블록으로 나타낸 것은 중간 데이터이며, 그 중간 데이터에 합성곱 연산을 연속해서 적용함
- 층이 깊어지면서 더 복잡하고 추상화된 정보가 추출됨
- 층이 깊어지면서 뉴런이 반응하는 대상이 단순한 모양에서 고급 정보로 변화됨

LeNET

- CNN의 원조
- 손글씨 숫자를 인식하는 네트워크로 1998년에 제안됨
- 신경망 구조 - 합성곱 계층과 풀링 계층을 반복하고 마지막으로 완전연결 계층을 거치면서 결과를 출력함

LeNet과 현재의 CNN차이

- 활성화 함수 - 현재는 ReLU를 사용, LeNet은 시그모이드 함수를 사용함
- LeNet은 서브샘플링을 하여 중간 데이터의 크기를 줄이는 반면 현재는 최대 풀링을 사용함

AlexNet

- 딥러닝이 주목 받도록 함
- 2012년 ISVRC에서 우승한 딥러닝 모델
- 딥러닝 연구와 응용의 폭발적인 성장을 촉발시킴

LeNet과 비교시 AlexNet의 차이

- 활성화 함수로 ReLU를 이용
- LRN이라는 국소적 정규화를 실시하는 계층을 이용
- 드롭아웃 사용
- LeNet과 AlexNet은 큰 차이가 없지만 컴퓨터 기술(빅데이터, GPU 등)의 진보로 딥러닝이 발전 됨

딥러닝
- 층을 깊게 한 심층 신경망
- 딥러닝의 특징과 과제
- 딥러닝의 가능성

층을 깊게 할 수록 성능 올라감

데이터 확장(회전, 이동 등) -> 미세한 변화로 이미지 갯수 늘려서 TEST(데이터 양 많아짐 -> 정확도 올라감

층을 깊게하고 커널의 사이즈 작게 -> 매개 변수 자체가 줄어들고 학습이 더 잘됨(성능 올라감)

층이 깊어 질 수록 -> CNN 1층 엣지 블록 형태 인식 , 층이 깊어질수록 객체 OBJECT 인식함

정보를 계층적으로 전달 가능, 문제 계층적으로 분해 >>> 학습 성능 올라감, 복잡한 문제 해결 할 수 있다.

*딥러닝 2번의 위기
- 마빈 위스키 XOR 문제 학습 누가 시키는가.
- 층을 깊게 할 수록 기울기 손실 문제로 학습 안됨 -->시그모이드 등으로 해결

이미지넷 딥러닝으로 우승 후 주목 다시 받게됨
이미지넷 2015 인간의 수준 까지 따라잡음 (깊이 150층)

VGG
깊이 경쟁 계속됨

GoogLeNet
- 인셉션 구조 만들어 냄

사물 검출될때만 CCTV 저장
-> RCNN 사용

분할
- 픽셀 수준으로 분류
- FCN
완전연결층과 다르게 FCN에서 완전영역 계층도 CNN으로 만듬

사진캡쳐(NIC)
사진을 보고 글 생성, multi modal 처리

이미지 화풍 변경하여 그림 생성

DCGAN
-> 생성자와 식별자 사용하여 과제를 계속 만들어내고 식별함
-> 가짜 이미지 생성
-> 두개를 경쟁시키면서 DEEPFAKE 가 등장

자율 주행

강화 학습 - 더 나은 보상을 받는 쪽으로 에이전트는 스스로 학습함
-> 환경에서 주는 보답을 학습해 감 -> 최적화
-> 에이전트 스스로 학습

DQN (Deep Q-Network)
- 게임 마다 설정 바꾸지 않고 강화학습 알아서 학습

ai 모델 중 트랜스퍼머 가 성능 좋음

cnn 이미지 쪽 탁월
트랜스 퍼머가 cnn을 능가함
비전 트렌스포머 -> 현 상황에서 제일 좋은 모델

ResNet
- 층이 너무 깊으면 학습이 잘 안된다. 오히려 성능 떨어짐
-> 이 문제 해결 하기 위해 스킵 연결 도입함
>> 성능 향상 됨, 스킵 연결 많이 사용됨

클라우드가 아니라 디바이스에 딥러닝 넣기 시작
mobileNet 등장
컴퓨터 리소스 비교

mobileNet
- 기존의 CNN에 비해 매개변수 수와 계산 요구 사항을 크게 줄임
디바이스에 올려야해서 크기를 점점 줄임

AutoML
- 기계 학습 모델 개발 작업을 자동화하는 프로세스

사이즈 줄이면 정확도 약간 낮아져도 근접한 모델 만들어 가는중

딥러닝의 고속화 - ResNet, 대량 연산을 수행해야 하니까 gpu몇개 하다가 alexNet은 gpu 2개로
학습의 복수의 gpu, 여러 기계로 분산하여 학습

딥러닝에서의 처리 시간 요소
컴플루션 계산 -> 많은 시간 소요 하고 있다.
- 합성곱 계층에서 이뤄지는 연산 고속으로 계산 어떻게 하느냐가 딥러닝의 과제

GPU 컴퓨팅 (일반화시킨 범용 수치)
- cpu 폰노이만, gpu ALU를 늘여놓은..
- ALEXNET으로 비교

분산 학습
- 딥러닝 학습을 수평 확장하자는 아이디어
- GPU 분산시켜 학습시키는 아키텍처 -> 구글의 텐서플로와 마이크로 소프트의 CNTK
- GPU 100개 사용하면 하나일때보다 56배 빨라짐

ai 모델중 앞으로 산업 구조 바꿀 것 예상
1. on디바이스(모바일, 가전, 비주얼 디스플레이, 로봇)
2. ai 로봇
3. 밀리터리
등등

chatGpt 수익이 별로 안됨. 운영, 전력 소비에 비해
ai가 더 활성화 하기 위해서 산업으로 뛰어들어야 함

김상옥

[24]:
```
import numpy as np
import matplotlib.pylab as plt
```

4차원 배열

np.random.rand(10, 1, 28, 28)

- 채널 1개인 28*28행렬 데이터가 10개

[25]:
```
x = np.random.rand(10, 1, 28, 28)
print("x shape --->", x.shape)
```

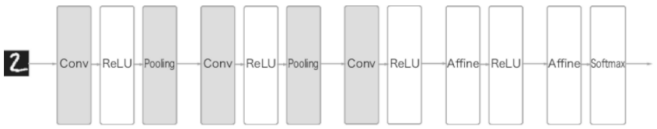
```
x shape ---> (10, 1, 28, 28)
```

[26]:
```
print("채널이 1개인 28*28 행렬 데이터 5번째")
print(x[5, :, :5, :5])
```

```
채널이 1개인 28*28 행렬 데이터 5번째
[[[0.22011433 0.94794482 0.99168345 0.88066347 0.580012  ]
  [0.36358243 0.39844021 0.52710818 0.45258772 0.16599453]
  [0.1895279  0.39632157 0.52499837 0.77297128 0.34995014]
  [0.71225876 0.957674   0.73093668 0.31483177 0.12790868]
  [0.0453436  0.16588091 0.64000391 0.18966794 0.69761601]]]
```

■ CNN 구조

- 합성곱 계층과 풀링 계층이 추가됨
- 마지막 출력 계층에서는 Affine – Softmax 조합을 그대로 사용



CNN - im2col 함수

- 4차원 행렬을 2차원으로 바꾸어줌

[27]:
```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    # 입출력 자료를 가져옴
    N, C, H, W = input_data.shape

    # 그 행렬과 필터를 계산했을 때 나오는 사이즈를 계산
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1

    print("out : ", out_w, out_h)

    # 패딩 적용
    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')

    # 출력 행렬 사이즈만큼 0으로 초기화
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))


    for y in range(filter_h):
        # 현재 y번째에서 y(열)의 범위 지정
        y_max = y + stride*out_h
        for x in range(filter_w):
            # 현재 x번째에서 x(행)의 범위 지정
            x_max = x + stride*out_w

            # 위에서부터 현재 열, 행부터 범위까지만큼 값을 col에 적용
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    # col를 2차원으로 변경
    # transpose - 꼬인 인덱스 순서로 변경? -> reshape - 차원 변경
    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
    return col
```

im2col함수 사용 예시

[28]:
```
x1 = np.random.rand(1, 1, 4, 4)
col1 = im2col(x1, 3, 3, stride=1, pad=0)
print(x1.shape, " -------> ",col1.shape)
```

```
out :  2 2
(1, 1, 4, 4)  ------->  (4, 9)
```

[29]:
```
x2 = np.random.rand(10, 3, 7, 7)
col2 = im2col(x2, 5, 5, stride=1, pad=0)
print(x2.shape, " -------> ",col2.shape)
```

```
out :  3 3
(10, 3, 7, 7)  ------->  (90, 75)
```

[30]:
```
x3 = np.random.rand(10, 3, 7, 7)
col3 = im2col(x3, 5, 5, stride=1, pad=0)
print(x3.shape, " -------> ",col3.shape)
```

```
out :  3 3
(10, 3, 7, 7)  ------->  (90, 75)
```

[31]:
```
x4 = np.random.rand(10, 3, 7, 7)
col4 = im2col(x4, 5, 5, stride=3, pad=5)
print(x4.shape, " -------> ",col4.shape)
```

```
out :  5 5
(10, 3, 7, 7)  ------->  (250, 75)
```

CNN에서 Pooling의 역할

최대값만 가져옴 (가중치 x)

크기를 줄일 수 있다

SimpleConvNet을 위한 함수 설정

[32]:
```
def numerical_gradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)

        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)

        x[idx] = tmp_val # 값 복원
        it.iternext()

    return grad
```

[33]:
```
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    # 훈련 데이터가 원-핫 벡터라면 정답 레이블의 인덱스로 변환
    if t.size == y.size:
        t = t.argmax(axis=1)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size


def softmax_loss(X, t):
    y = softmax(X)
    return cross_entropy_error(y, t)
```

[34]:
```
def relu(x):
    return np.maximum(0, x)


def relu_grad(x):
    grad = np.zeros(x)
    grad[x>=0] = 1
    return grad


def softmax(x):
    if x.ndim == 2:
        x = x.T
        x = x - np.max(x, axis=0)
        y = np.exp(x) / np.sum(np.exp(x), axis=0)
        return y.T

    x = x - np.max(x) # 오버플로 대책
    return np.exp(x) / np.sum(np.exp(x))
```

[35]:
```
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

        # 중간 데이터 (backward 시 사용)
        self.x = None
        self.col = None
        self.col_W = None

        # 가중치와 편향 매개변수의 기울기
        self.dW = None
        self.db = None

    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
        out_w = 1 + int((W + 2*self.pad - FW) / self.stride)

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T

        out = np.dot(col, col_W) + self.b
        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W

        return out

    def backward(self, dout):
        FN, C, FH, FW = self.W.shape
        dout = dout.transpose(0,2,3,1).reshape(-1, FN)

        self.db = np.sum(dout, axis=0)
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

        dcol = np.dot(dout, self.col_W.T)
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

        return dx


class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

        self.x = None
        self.arg_max = None

    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)

        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        self.x = x
        self.arg_max = arg_max

        return out

    def backward(self, dout):
        dout = dout.transpose(0, 2, 3, 1)

        pool_size = self.pool_h * self.pool_w
```

[36]:
```python
class SGD:
    def __init__(self, lr=0.01):
        self.lr =lr

    def update(self, params, grads):
        for key in params.keys():
            params[key] -= self.lr * grads[key]
```

[37]:
```python
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b

        self.x = None
        self.original_x_shape = None
        # 가중치와 편향 매개변수의 미분
        self.dW = None
        self.db = None

    def forward(self, x):
        # 텐서 대응
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        dx = dx.reshape(*self.original_x_shape)  # 입력 데이터 모양 변경(텐서 대응)
        return dx


class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None # 손실함수
        self.y = None    # softmax의 출력
        self.t = None    # 정답 레이블(원-핫 인코딩 형태)

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)

        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        if self.t.size == self.y.size: # 정답 레이블이 원-핫 인코딩 형태일 때
            dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
```

[38]:
```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import pickle
import numpy as np
from collections import OrderedDict


class SimpleConvNet:
    """단순한 합성곱 신경망

    conv - relu - pool - affine - relu - affine - softmax

    Parameters
    ----------
    input_size : 입력 크기 (MNIST의 경우엔 784)
    hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 (e.g. [100, 100, 100])
    output_size : 출력 크기 (MNIST의 경우엔 10)
    activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
    weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
        'relu'나 'he'를 지정하면 'He 초깃값'으로 설정
        'sigmoid'나 'xavier'를 지정하면 'Xavier 초깃값'으로 설정
    """
    def __init__(self, input_dim=(1, 28, 28),
                 conv_param={'filter_num':30, 'filter_size':5, 'pad':0, 'stride':1},
                 hidden_size=100, output_size=10, weight_init_std=0.01):
        filter_num = conv_param['filter_num']
        filter_size = conv_param['filter_size']
        filter_pad = conv_param['pad']
        filter_stride = conv_param['stride']
        input_size = input_dim[1]
        conv_output_size = (input_size - filter_size + 2*filter_pad) / filter_stride + 1
        pool_output_size = int(filter_num * (conv_output_size/2) * (conv_output_size/2))

        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * \
                            np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
        self.params['b1'] = np.zeros(filter_num)
        self.params['W2'] = weight_init_std * \
                            np.random.randn(pool_output_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * \
                            np.random.randn(hidden_size, output_size)
        self.params['b3'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b1'],
                                           conv_param['stride'], conv_param['pad'])
        self.layers['Relu1'] = Relu()
        self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
        self.layers['Affine1'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W3'], self.params['b3'])

        self.last_layer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

    def loss(self, x, t):
        """손실 함수를 구한다.

        Parameters
        ----------
        x : 입력 데이터
        t : 정답 레이블
        """
        y = self.predict(x)
        return self.last_layer.forward(y, t)
```

```
def accuracy(self, x, t, batch_size=100):
    if t.ndim != 1 : t = np.argmax(t, axis=1)

    acc = 0.0

    for i in range(int(x.shape[0] / batch_size)):
        tx = x[i*batch_size:(i+1)*batch_size]
        tt = t[i*batch_size:(i+1)*batch_size]
        y = self.predict(tx)
        y = np.argmax(y, axis=1)
        acc += np.sum(y == tt)

    return acc / x.shape[0]

def numerical_gradient(self, x, t):
    """기울기를 구한다 (수치미분) .

    Parameters
    ----------
    x : 입력 데이터
    t : 정답 레이블

    Returns
    -------
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['W1']、grads['W2']、... 각 층의 가중치
        grads['b1']、grads['b2']、... 각 층의 편향
    """
    loss_w = lambda w: self.loss(x, t)

    grads = {}
    for idx in (1, 2, 3):
        grads['W' + str(idx)] = numerical_gradient(loss_w, self.params['W' + str(idx)])
        grads['b' + str(idx)] = numerical_gradient(loss_w, self.params['b' + str(idx)])

    return grads

def gradient(self, x, t):
    """기울기를 구한다(오차역전파법).

    Parameters
    ----------
    x : 입력 데이터
    t : 정답 레이블

    Returns
    -------
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['W1']、grads['W2']、... 각 층의 가중치
        grads['b1']、grads['b2']、... 각 층의 편향
    """
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.last_layer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Conv1'].dW, self.layers['Conv1'].db
    grads['W2'], grads['b2'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W3'], grads['b3'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

    return grads

def save_params(self, file_name="params.pkl"):
    params = {}
    for key, val in self.params.items():
        params[key] = val
    with open(file_name, 'wb') as f:
        pickle.dump(params, f)

def load_params(self, file_name="params.pkl"):
    with open(file_name, 'rb') as f:
```

```
def save_params(self, file_name="params.pkl"):
    params = {}
    for key, val in self.params.items():
        params[key] = val
    with open(file_name, 'wb') as f:
        pickle.dump(params, f)

def load_params(self, file_name="params.pkl"):
    with open(file_name, 'rb') as f:
        params = pickle.load(f)
    for key, val in params.items():
        self.params[key] = val

    for i, key in enumerate(['Conv1', 'Affine1', 'Affine2']):
        self.layers[key].W = self.params['W' + str(i+1)]
        self.layers[key].b = self.params['b' + str(i+1)]
```

이전 과제에 CNN을 적용하여 실습해보기

```
[39]: import sys, os
      sys.path.append(os.pardir)
      import numpy as np
      from dataset.mnist import load_mnist
      import matplotlib.pyplot as plt
      from collections import OrderedDict

[40]: (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

[42]: # 100, [15, 15, 15], 0.011

      # 287, [15, 15, 15, 15, 15], 0.015


      iters_num = 10000
      batch_size = 500
      #hidden_size = 15
      hidden_size = [15, 15, 15, 15, 15]
      learning_rate = 0.015

      # 기울기 최적화 함수들
      sgd = SGD()

      train_loss_list = []
      train_acc_list = []
      test_acc_list = []

      train_size = x_train.shape[0]
      iter_per_epoch = max(train_size / batch_size, 1)

      network = SimpleConvNet(input_size=784, hidden_size=hidden_size, output_size=10)

      for i in range(iters_num):
          batch_mask = np.random.choice(train_size, batch_size)
          x_batch = x_train[batch_mask]
          t_batch = t_train[batch_mask]

          grad = network.gradient(x_batch, t_batch) # 오차역전파법 방식

          # 일반 경사
          #for key in ('W1', 'b1', 'W2', 'b2'): network.params[key] -= learning_rate * grad[key]

          # SGD 확장, 모멘텀 확장, AdaGrad 확장
          sgd.update(network.params, grad)
          #momentum.update(network.params, grad)
          #adaGrad.update(network.params, grad)


          loss = network.loss(x_batch, t_batch)
          train_loss_list.append(loss)

          if i % iter_per_epoch == 0:
              train_acc = network.accuracy(x_train, t_train)
              test_acc = network.accuracy(x_test, t_test)
              train_acc_list.append(train_acc)
              test_acc_list.append(test_acc)
              print("%2d 번째 학습(오류) --> train_acc : %.2f%%, test_acc : %.2f%%" % (len(train_acc_list), train_acc*100, test_acc*100))

      print("최종 학습률 --> train_acc : %.2f%%, test_acc : %.2f%%" % (train_acc_list[-1]*100, test_acc_list[-1]*100))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[42], line 22
     19 train_size = x_train.shape[0]
     20 iter_per_epoch = max(train_size / batch_size, 1)
---> 22 network = SimpleConvNet(input_size=784, hidden_size=hidden_size, output_size=10)
     24 for i in range(iters_num):
     25     batch_mask = np.random.choice(train_size, batch_size)

TypeError: SimpleConvNet.__init__() got an unexpected keyword argument 'input_size'
```

오류...

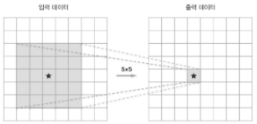
딥러닝 8장

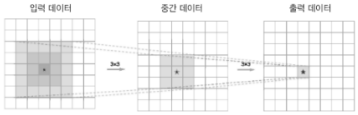
- 딥러닝: 층을 깊게 한 심층 신경망

기존보다 더 깊은 신경망으로 보다 복잡한 문제를 해결할 수 있는 딥러닝

- 층을 깊게 하는 이유

- 신경망의 매개 변수가 줄어든다

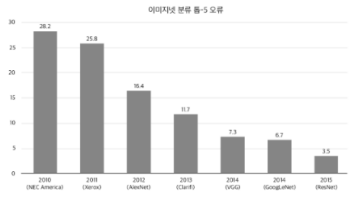




- 5x5의 매개변수는 25개 - 3x3은 18개로 매개 변수를 줄일 수 있다
- - 매개변수가 줄어든다는 점
  - 이는 연산에 들어갈 비용을 줄일 수 있다는 뜻이다

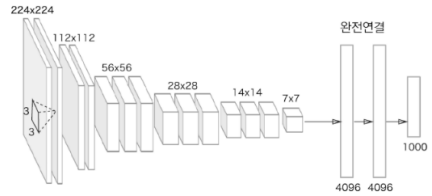
신경망 정확도를 높이기 위한 기술

- 앙상블 학습
- 학습률 감소
- 데이터 확장
- - 인위적으로 이미지의 회전이나 이동 등 미세한 변화를 주어 이미지의 개수를 늘린다



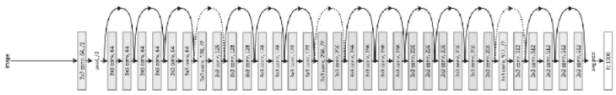
ILSVRC대회에서 AlexNet이 나타나면서 딥러닝의 시대를 열었다

VGG : 3x3 필터를 사용하여 합성곱 계층을 연속으로 거친다



GoogLeNet : (아래 사진) 각 사각형이 CNN계층, Pooling계층을 나타냄 - 세로/가로로 깊다는 점이 특징이다

ResNet : VGG신경망을 기반으로 스킵 연결을 도입하여 층을 깊게 하였다



딥러닝의 발전

- 빅데이터(학습 데이터?), 네트워크의 발전

- 하드웨어의 발전 -> 딥러닝 고속화

- - 기존 연산에 특화된 GPU를 AI연산에 활용하면서 AI학습이 매우 빠르게 진행되게 되었다(엔비디아 덕상)

- 분산 학습

딥러닝의 활용

- 딥러닝은 이미지에서 사물을 구분, 분할, 자연어 처리, 이미지 생성, 자율 주행 등... 매우 다양한 분야에서 활용될 수 있다

강화학습

- 더 좋은 보상으로 스스로 학습하는 방법

DQN

- Q학습에서 최적 행동 가치 함수로 행동을 정함..?