인공지능개론 정리노트#4

201904022 김상옥

202284026 안정빈

질문1. 하이퍼파라미터 값 탐색을 효율적으로 하는 방법과 이유는? (6장)

김상옥: 학습 데이터를 줄이고, 다양한 하이퍼파라미터 값으로 여러 개의 신경망 학습을 진행하여, 적절한 값을 찾아 나가는 방법이 좋지 않을까? 학습 데이터 전체를 하이퍼파라미터 하나의 경우로 테스트하는 것 보다 작은 학습 데이터로 여러 번 하는 게 효율적이고 또 괜찮은 값들의 리스트를 얻고, 그 리스트를 바탕으로 다시 학습 데이터에 적용해보면서 찾아가는 방법이 효율적이라고 생각해.

안정빈: 의견에 동의해. 학습 데이터 전체를 사용하는 것보다 작은 데이터 셋을 사용하여 여러 번 학습하기 때문에 전체적인 계산 비용을 줄일 수 있어. 특히, 대규모 데이터 셋을 다루는 경우 효과가 클 것 같아. 그리고 학습 데이터 전체를 사용하는 경우 모든 하이퍼파라미터 조합에 대해 성능을 평가해야 하기 때문에 탐색 속도가 느려질 수 있어. 반면, 작은 데이터 셋을 사용하면 탐색 속도를 높일 수 있어. 이러한 측면에서 효율성을 높일 수 있지만 주의해야 할 점도 있지 않을까?

김상옥: 적절한 학습 데이터의 수를 조절하는 게 중요할 거 같아. 너무 작은 데이터면 의미가 없는 결과가 나타날 수 있을 거 같고, 그렇다고 너무 큰 데이터라면 시간에 효율이 나오지 않을 거 같아. 결국 적절한 신경망 구조나 함수들 또 적절한 하이퍼파라미터 그리고 그 하이퍼파라미터를 구하기 위한 적절한 방법... 이 학습이라는 게 결국 정답이 없는 것이기에 최적의 값들을 찾는 것이 신경망에 매우 중요할 거 같아.

안정빈: 맞아. 데이터의 수를 결정하는 것 중요한 것 같아. 작은 데이터는 편향된 결과를 초래할 가능성이 있어. 이는 최적의 하이퍼파라미터 값을 찾는 데 영향을 미칠 것 같고 모델 성능 평가하는데 어려운 것 같아. 이러한 문제점을 고려해서 최적의 방법을 선택해야 하는 것 같아.

김상옥: 나중에는 인공지능 신경망의 하이퍼피라미터를 최적화해주는 인공지능도 나올 거 같아. 혹은 그런 프로그램이 나타난다거나. 어쨌든 그만큼 신경망에서 하이퍼피라미터가 중요하다는 의미이고, 하이퍼피라미터의 값을 효율적으로 찾는 것도 중요하다는 의미겠지. 우리가 말했듯이 작은 데이터의 그룹을 여러 개 나누어서 찾는 것이 그나마 효율적인 거 같아.

결론 : 하이퍼파라미터 값 탐색은 신경망을 최적화하는데 중요한 과정입니다. 작은 데이터의 그룹을 여러 개 나누어서 최적 값을 찾는 방법을 통해 계산 비용을 줄이고 탐색 속도를 높일 수 있습니다.

질문2. 학습 데이터 정확도만 높거나 시험 데이터 정확도와의 차이가 크면 안 되는 이유(6장)

안정빈: 학습 데이터 정확도만 높거나 시험데이터 정확도와 차이가 크면 overfitting이 발생하는 것으로 알고 있어. 오버 피팅이란 신경망이 훈련 데이터에만 지나치게 적응되어 그 외의 데이터에는 제대로 대응하지 못하는 상태라고 알고 있어. 오버 피팅은 어떤 문제를 발생시킬까?

김상옥: 시험 데이터와 정확도가 차이가 난다는 것은 그만큼 실제 문제에 대해서 안정적이지 않다는 의미라고 해석할 수 있겠다. 결국 학습을 하는 이유나 오버 피팅을 방지하는 이유는 학습하지 못한 데이터에 대해서도 안정적인 결과를 얻기 위 함이니까. 사람을 구분하는 AI인데 그림에 있는 사람까지 구분해버리고 어떤 처리를 한다면 어떠한 문제가 발생할 수 있으니까. 실사용에서 안정적인 결과를 위해 오버 피팅을 막아야 할 거 같아.

안정빈: 맞아. 학습 데이터와 유사한 데이터에 대해서는 좋은 성능을 보이고 새로운 데이터에서는 성능이 떨어져. 학습 데이터의 작은 변화에도 민감하게 반응해서 성능도 크게 변동할 수 있고, 모델이 학습데이터의 특정 패턴을 많이 학습하면 모델이 예측하기 오히려 어려워져. 따라서 학습 데이터 정확도와 시험 데이터 정확도도 함께 고려해야 한다고 생각해. 과적합을 방지하는 방법도 있을까?

김상옥 : 이 부분에서는 학습 데이터 자체도 중요하다고 생각해. 최대한 다양한 경우의 질 좋은 학습 데이터가 좋은 학습을 유도할 수 있을 거야. 좋은 강의를 듣고 시험을 보는 것처럼 신경망의 학습 데이터도 다양하고 좋은 데이터를 선별해서 학습하는 것도 중요할 거 같다. 그리고 다음 장에서 배우는 CNN도 이런 이유를 포함해서 사용하는 것이지 않을까? 결국 더 좋고 효율적인 것을 추구하는 것이니까.

안정빈 : 데이터 증강하여 더 다양한 데이터를 경험하도록 하는 방법이 있었지. 그리고 시험 데이터 성능이 더 이상 향상되지 않으면 학습을 중단해서 과적합을 방지하는 방법도 있는 것 같아. 이렇게 과적합을 방지하는 방법들이 있지만. 오버 피팅이 발생하기전에 방지하는 것도 좋은 방법일 것 같아. 학습데이터와 시험 데이터의 정확도를 잘 조절해야 할 것 같아.

김상옥: 그런 문제를 위해서도 수업시간에 나온 드롭 아웃 같은 다양한 해결방법의 원리를 알고 적용하는 것이 중요한 거 같아. 이것도 결국에는 여러가지 복잡한 요소들이 조화롭게 이루어져야 오버 피팅이 발생하지 않는 좋은 학습 결과를 얻을 수 있을 거야. 이런 내용을 말할수록 신경망을 하나 만든다는 것이 얼마나 복잡하고 체계적인 구조 속에서 학습이 이루어져야 하는 지 조금이나마 느껴지는 거 같아.

결론: 학습 데이터 정확도만 높거나 시험 데이터 정확도와의 차이가 크면 오버 피팅이 발생하게 됩니다. 즉, 학습 데이터, 시험 데이터의 정확도를 고려하여 학습시켜야 합니다. 드롭 아웃, 하이퍼파라미터 값을 조절하는 등 방법으로 오버 피팅을 해결할 수 있습니다.

질문3. 합성곱 신경망이 대뇌의 시각 피질과 어떤 연결성이 있어서 발견하고 개발하게 된 걸까? (7장)

김상옥: CNN의 역사에서 사람의 대뇌 시각 피질에서 아이디어를 얻어 개발했다고 했는데 그냥 봤을 때는 이게 왜 CNN과 연결이 되었는지 이해가 잘 안가는 부분이 있어. 국부수용장이라는 모델을 모방했다고 말했는데 이 국부수용장이 무엇을 의미하고 사람의 시각 피질과 CNN은 어떤 연결성을 가지고 있을까?

안정빈: 국부 수용장은 시각 정보 처리 시스템에서 특정 영역에만 민감하게 반응하는 뉴런 집단을 의미해. CNN 모델은 국부 수용장 개념을 활용하여 시각 정보를 효율적으로 처리하고 분석한다고 해. CNN의 필터는 국부 수용장을 나타내고 입력 이미지의 특정 영역에 적용되어 특정한 패턴을 인식해. 시각 피질과 CNN 모델은 모두 계층적 구조를 가지고, 점점 더 추상화된 정보를 처리하는 방식으로 작동해. 그리고 모두 국부 수용장 개념을 활용하여 시각 정보를 처리한다는 공통점이 있어.

김상옥: 국부 수용장에 의미가 특정 영역에만 반응하는 뉴런 집단라는 것이 어떤 의미일까? 사람이 무언가를 보고 인지하는 과정은 빛이라는 정보가 감각기관 눈을 통해서 뇌에서 인지하는 것이 시각이고, 이 과정에서 뉴런들이 정보들을 전달하는 역할이니까 국부 수용장은 무언가를 볼 때 특정 영역에만 반응하는 것이지 않을까? 다시 말해서 사람이 사과를 보면 사과라는 정보가 뇌에 전달되는 것이 아니라 보고있는 장면의 여러가지 요소들이 각 뉴런들을 활성화하고, 그 정보들의 합집합을 통해서 사물을 인지한다는 의미라고 생각하는데 너의 생각은 어때? 이런 방식에서 아이디어를 얻어 CNN을 만들게 된 것이고.

안정빈: 너가 말한 예시처럼 뉴런들이 특정 범위안에 있는 시각 자극에만 반응하는 것이 맞는 것 같아. 주요 특징에만 집중하여 정보 처리 효율성을 높여주고 주요특징을 강조하여 추출해 주게 되는 것 같다는 생각이 들어. 앞서 말한 것처럼 계층적 구조를 가지고 있는데 어떠한 과정을 가지게 되는 것일까?

김상옥: CNN을 보면 행렬과 필터를 통해서 결과를 출력하고, 그런 것을 반복하여 결국에는 n차원의 데이터를 신경망에 입력하게 되는 형식인거 같아. 패턴이나 특정 부분을 구별하기 위해서 필터를 많이 적용하다보면 그만큼 계층적인 구조가 형성될 것이라고 생각해. 물론 적으면 작은 계층이 형성되고. 즉 행렬과 필터의 연산이 하나의 계층속에서 일어나는 계산이라면 계층에서 나온 출력이 다시 필터와 계산되어 하나의 계층이 또 형성되는 식으로 적절한 계층을 만들지 않을까?

안정빈 : 그런 계층이 맞는 것 같아. 계층적 구조와 국부 수용장 개념을 활용해서 결국 시각 피질에서 전체 뉴런이 아닌 특정 뉴런만이 활성화되는 것을 발견하였고 일정 범위안의 자극에만 활성화되는 근접 수용 영역을 가진다는 것을 알게 되어서 개발을 하게 되었다는 것을 알게 됐어.

결론: 사람은 사물을 인지할 때 빛을 통해 들어온 정보가 감각 기관을 지나 뇌에서 인지한다. 그 과정에서 뉴런들이 활성화되는데 뉴런들의 집합이 한번에 활성화 되는 것이 아닌 국부적으로 활성화되어서 하나의 사물, 장면을 인지한다. 그것을 국부수용장이라 부르며 CNN은 이 과정에서 아이디어를 얻어 인공지능에서도 국부수용장의 원리를 적용하여 개발하게 되었다.

질문4. CNN으로 얻을 수있는 이점(효과)(7장)

김상옥: CNN에 대해서 간단하게 살펴봤었는데 하나의 행렬을 필터를 통해서 결과값을 나타내고, 그 값을 신경망에 입력해주면서 활용한다고 배웠어. 또 내가 듣는 디지털 영상처리 수업에서는 잠깐 나온 이야기로는 CNN이 이미지에 많이 사용된다고 하였는데 CNN이 이런 부분에 있어서 분명한 강점이 있을 거라고 생각해. CNN을 사용하는 이유가 분명 있기 때문에 배우고 많이 사용할 것인데 CNN을 하면 얻을 수 있는 이점(효과)는 무엇이 있을까?

안정빈: 맞아. 나도 CNN은 이미지 인식, 분류 등에 우수한 성능을 보여준다고 알고 있어. CNN을 사용하면 이미지 인식 및 분석작업에서 높은 정확도를 제공하고 이미지에서 중요한 특징을 효과적으로 추출할 수 있어. 이는 객체 인식, 이미지 분류 등의 작업에서 중요한 역할을 한다고 해. 그리고 다양한 이미지 패턴을 학습하고 정확한 예측을 수행할 수 있어. CNN은 다양한 분야에서 실제로 사용되고 있다 고하는데 어떤 분야들에서 활용되고 있을까?

김상옥: CNN이 이미지에 특징을 효과적으로 확인할 수 있기 때문에 이미지에서 많이 활용되는 거구나. CNN이 어떤 패턴이나 특징을 알아내는데 좋다면 정말 많은 분야에서 활용될 수 있지 않을까? 특히 코드를 작성하는 것도 디자인 패턴같은 정형화된 코드 작성 패턴이 많이 있으니 코드 작성 AI에서도 CNN이 활용될 수 있을 거 같아. 그리고 코드와 비슷하게 우리가 말하는 것도 일종의 패턴이 있다면 지금 GPT나 Gemine같은 생성 AI에서도 CNN이 활발하게 활용될 거라고 생각해. 대부분의 경우에서 패턴이 나타나고, 그런 점에서는 CNN을 사용하는 이유를 어느정도 이해할 수 있는 거 같아. 그럼 반대로 CNN을 사용하지 않아야 할 경우나 그런 분야가 있을까?

안정빈: CNN은 이미지 인식 분야에서 뛰어난 성능을 발휘하지만 사용하지 않아야 하는 경우가 있을 거야. 데이터가 부족하여 학습 데이터 확보가 어려운 경우나, 의료 영상 진단 등 설명 가능한 결과가 필요한 분야, 예술 작품을 제작하거나 디자인 등 창의성이 필요한 작업에는 CNN을 사용할 수 없다고 생각해. 신뢰성, 윤리적 문제 등 다양한 문제들이 생기지 않을까?

김상옥: 강력한 힘을 가진 AI를 무분별하게 만들고 학습하다보면 어떤 문제를 초래할 지 모르고, 그런 부분이 인공지능의 블랙박스라고도 불리니까 AI를 만든다고 할 때는 사회적으로 안전을 고려하면서 제작해야 할거야. 이런 규칙이 기반이 되어 AI를 제작해야 한다고 생각해. 정말 영화처럼 AI가 인간을 지배할 수도 있는게 그냥 나오는 말은 아니니까. 정리해보면 이미지같은 패턴을 알아내는 점에서 CNN은 이점이 있다는 점을 알 수 있었고, 그런 점에서 CNN도 AI학습에 분명 도움이 되겠지만 그것도 상황에 따라 다를 것이고 아직은 생각나지는 않는데 CNN이 오히려 독이되는 경우도 있을 수 있겠지. 이 토론을 통해 너가 생각하는 CNN의 이점은 뭐야?

결론 : CNN은 이미지의 특징을 효과적으로 추출하고 분류하기 때문에 다양한 이미지 인식 작업에서 높은 정확도를 보여주고 중요한 특징을 추출 할 수 있습니다. 이러한 이점들로 CNN을 다양한 분야에서 활용할 수 있다는 이점들이 또 생기는 것 같습니다. 자율 주행 자동차, AI, 개인 맞춤형 광고 등 에서 활용 됩니다. 저희 삶에 편리함을 더해 주는 것 같습니다.

안정빈

신경망 학습의 목적 : 손실 함수의 값을 가능한 낮추는 매개변수를 찾는것
최적화 : 매개변수 최적값 찾는 것
최적의 매개 변수 찾는 방법 : 매개변수의 기울기(미분)을 구해 기울어진 방향으로 매개변수 값을 갱신하는 일을 반복하여 최적의 값 다가감 -> **SGD, 확률적 경사 하강법**

확률적 경사 하강법의 단점 : 비등방성 함수

단점 개선해주는 3가지 방법

- 모멘텀
- AdaGrad
- Adam

모멘텀 (= 운동량)

- 인스턴스 변수 v 가 물체의 속도임, v는 초기화 때는 아무 값도 담지 않음
- update()가 처음 호출 될 때 매개변수와 같은 구조의 데이터를 딕셔너리 변수로 저장

```python
class Momentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = self.momentum*self.v[key] - self.lr*grads[key]
            params[key] += self.v[key]
```

AdaGrad

- '각각의' 매개변수에 '맞춤형' 값을 만들어줌
- 주의할 점은 1e-7 작은 값을 더하는 부분(0으로 나누는 사태를 막아줌

AdaGrad를 사용한 최적화 문제 해결

- 최솟값을 향해 효율적으로 움직
- • y축 방향은 기울기가 커서 처음에는 크게 움
- • 큰 움직임에 비례해 갱신 정도도 큰 폭으로 작아지도록
- • Y 축 방향으로 갱신 강도가 빠르게 약해지고, 지그재그 움직임이 줄어듬

```python
class AdaGrad:
    def __init__(self, lr=0.01):
        self.lr = lr
        self.h = None

    def update(self, params, grads):
        if self.h is None:
            self.h = {}
            for key, val in params.items():
                self.h[key] = np.zeros_like(val)

        for key in params.keys():
            self.h[key] += grads[key] * grads[key]
            params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

Adam : 모멘텀과 AdaGrad 기법을 융합 , 하이퍼 파라미터 '편향 보정'이 진행된다.

```python
class Adam:

    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t  = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)

        for key in params.keys():
            #self.m[key] = self.beta1*self.m[key] + (1-self.beta1)*grads[key]
            #self.v[key] = self.beta2*self.v[key] + (1-self.beta2)*(grads[key]**2)
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])

            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)

            #unbias_m += (1 - self.beta1) * (grads[key] - self.m[key]) # correct bias
            #unbisa_b += (1 - self.beta2) * (grads[key]*grads[key] - self.v[key]) # correct bias
            #params[key] += self.lr * unbias_m / (np.sqrt(unbisa_b) + 1e-7)
```

오버피팅 : 신경망이 훈련 데이터에만 지나치게 적응되어 그 외의 데이터에는 제대로 대응하지 못하는 상태

오버피팅이 일어나는 경우

- 매개변수가 많고 표현력이 높은 모델
- 훈련 데이터가 적음

```python
# coding: utf-8
import os
import sys

sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.multi_layer_net import MultiLayerNet
from common.optimizer import SGD

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

# 오버피팅을 재현하기 위해 학습 데이터 수를 줄임
x_train = x_train[:300]
t_train = t_train[:300]

# weight decay (가중치 감쇠) 설정
#weight_decay_lambda = 0 # weight decay를 사용하지 않을 경우
weight_decay_lambda = 0.1

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10,
                        weight_decay_lambda=weight_decay_lambda)
optimizer = SGD(lr=0.01) # 학습률이 0.01인 SGD로 매개변수 갱신

max_epochs = 201
train_size = x_train.shape[0]
batch_size = 100
```

가중치 감소 : 학습 과정에서 큰 가중치에 대해서는 그에 상응하는 큰 페널티를 부과하여 오버피팅을 억제하는 방법
드롭 아웃 : 뉴런을 임의로 삭제하면서 학습하는 방법, 훈련 때 은닉층의 뉴런을 무작위로 골라 삭제합니다.
삭제된 뉴런은 신호를 전달하지 않게 되고 훈련 때는 데이터를 흘릴 때 마다 삭제할 뉴런을 무작위로 선택하고, 시험 때는 모든 뉴런에 신호를 전달합니다.
단, 시험 때는 각 뉴런의 출력에 훈련 때 삭제 안 한 비율을 곱하여 출력합니다.

(드롭 아웃은 뉴런을 무작위로 선택해 삭제하여 신호 전달을 차단한다.)

```python
class Dropout:
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None
    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x* self.mask
        else:
            return x* (1.0-self.dropout_ratio)
    def backward(self, dout):
        return dout*self.mask
```

드롭 아웃 실험 결과

- 훈련 데이터와 시험 데이터에 대한 정확도 차이가 줄었음
- 드롭 아웃을 이용하면 표현력을 높이면서도 오버피팅을 억제 할 수있음

하이퍼파라미터 : 각 층의 뉴런 수 , 배치크기, 매개변수 갱신 시의 학습률과 가중치 감소 등
하이퍼파라미터 값을 적절히 설정하지 않으면 모델의 성능이 크게 떨어짐

검증데이터
지금 까지 데이터 셋을 훈련 데이터 와 시험 데이터 라는 두가지로 분리해 이용했었습니다.
하이퍼파라미터의 성능을 평가할 때는 시험 데이터를 사용하면 안됩니다.
하이퍼파라미터 조정용(전용 확인) 데이터 : 검증 데이터

- 훈련 데이터 - 매개변수 학습
- 시험 데이터 - (신경망의)범용 성능 평가
- 검증 데이터 - 하이퍼파라미터 성능 평가
- -> 오버피팅, 범용 성능 어느정도 인지 평가 가능

하이퍼파라미터 최적화

- 하이퍼 파라미터를 최적화 할 때의 핵심은 하이퍼파라미터의 '최적 값'이 존재하는 범위를 조금씩 줄여간다.
- 범위를 조금씩 줄이려면 우선 대략적인 범위를 설정하고 그 범위에서 무작위로 하이퍼파라미터 값을 골라낸 후,그 값으로 정확도를 평가합니다.
- 정확도를 잘 살피면서 이 작업을 여러번 반복하여 '최적 값'의 범위를 좁혀간다.
- -> 하이퍼파라미터 값 탐색은 최적 값이 존재할 법한 범위를 점차 좁히면서 하는 것이 효과적이다.

합성곱 신경망(CNN)

- 이미지 인식과 음성 인식 등 다양한 곳에서 사용된다.
- CNN에서는 합성곱 계층과 폴링 계층이 추가됩니다.
- CNN에서는 합성곱 계층의 입출력 데이터를 **특징 맵** 이라고 합니다.

합성곱 계층의 입력 데이터 : 입력 특징 맵
합성곱 계층의 출력 데이터 : 출력 특징 맵

합성곱 연산

- 이미치 처리에서 말하는 필터 연산에 해당한다.
- 합성곱 연산은 입력 데이터에 필터를 적용합니다.
- 필터(=커널)
- 필터의 윈도우를 일정 간격으로 이동해가며 입력 데이터에 적용합니다.
- 여기서 말하는 윈도우는 3X3
- 단일 곱셈-누산 : 입력과 필터에서 대응하는 원소끼리 곱한 후 그 총합을 구합니다.
- 그리고 그 결과를 출력의 해당 장소에 저장합니다.
- -> 이 과정을 모든 장소에서 수행하면 합성곱 연산의 출력 완성

패딩

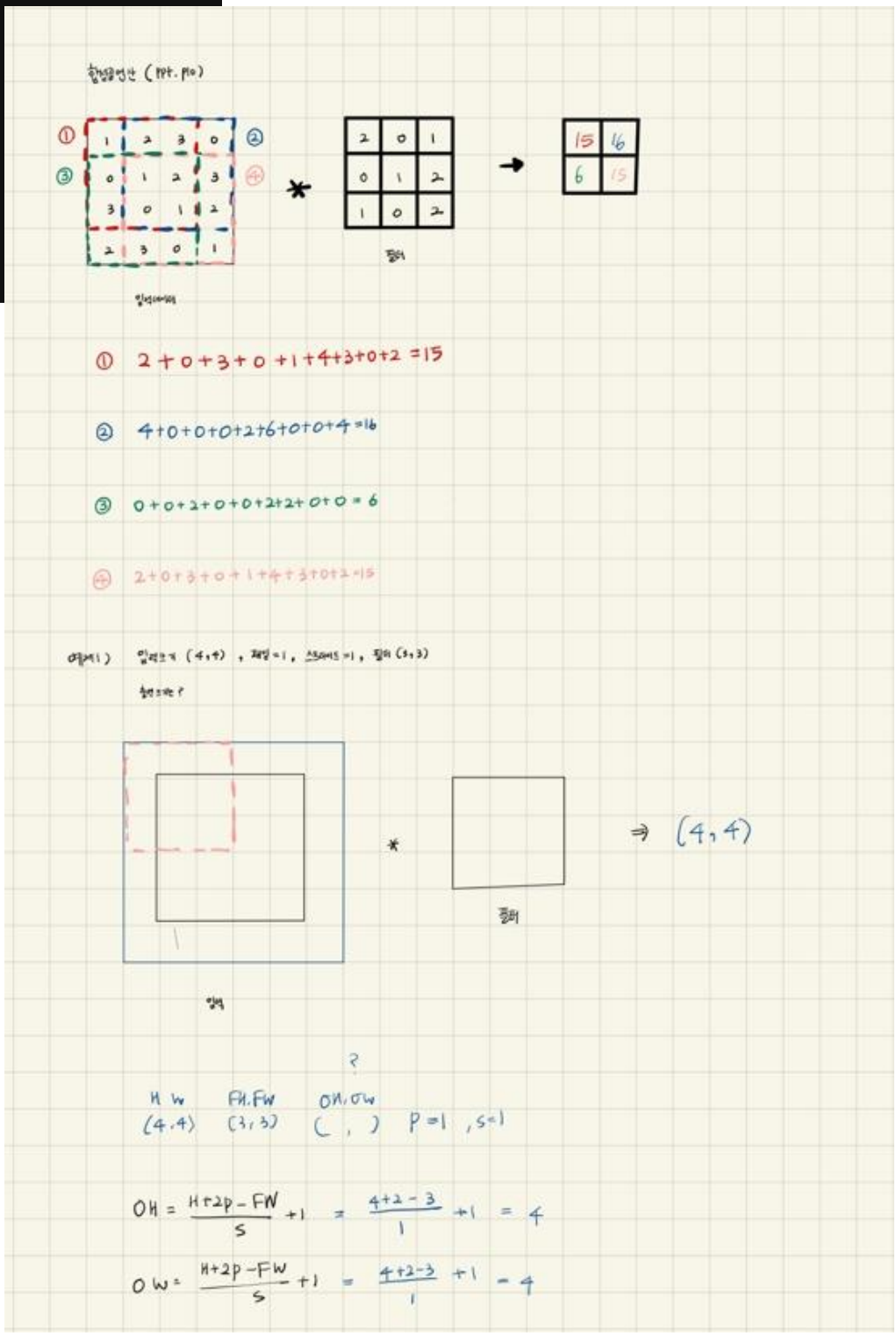
- 합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값(0)으로 채우기도 합니다.
- 주로 출력 크기를 조정할 목적으로 사용합니다.

스프라이드

- 필터를 적용하는 위치의 간격

풀링 계층

- 풀링은 세로, 가로 방향의 공간을 줄이는 연산
- 최대 풀링 : 최댓값을 구하는 연산
- 평균 풀링 : 평균값을 구하는 연산
  풀링 계층의 특징
- 학습해야 할 매개변수가 없다.
- 채널 수가 변하지 않는다.
- 입력의 변화에 영향을 적게 받는다.



김상옥

오버피팅

- 한쪽으로 치우쳐진 학습으로 범용성이 떨어진 것

오버피팅의 발생

- 학습 데이터와 시험 데이터 정확도의 차이가 크다
- 이는 학습 데이터에만 유의미한 결과를 보이고, 실제 문제에 대해서는 다른 답을 제시한다는 의미

수업 자료에 있는 코드

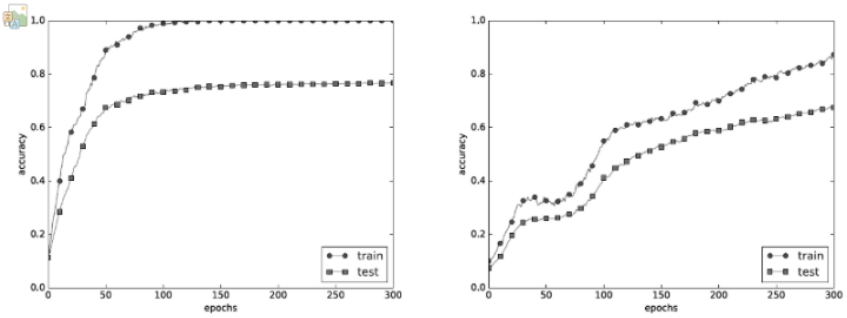
.0]:
```python
class Dropout:
    def __init__(self, dropout_ratio=0.5) :
        self.dropout_ratio = dropout_ratio
        self.mask = None

    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np. random. rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout * self.mask
```

드롭아웃

- 필요없는 뉴런을 선별하고 삭제하여 학습할 때 효과를 높인다



왼쪽은 둘다 높은 것 처럼 보이지만 학습과 시험의 정확도 차이가 크다 - 오버피팅

오른쪽은 둘이 적당한 간격으로 나타나 오버피팅을 어느정도 해소한 상태이다

[ ]:

하이퍼피라미터 단계

1. 값 설정
2. 값을 무작위로 추출
3. 학습하고 정확도 평가
4. 반복하여 하이퍼피라미터 값의 범위를 좁힌다

[138]:
```
import numpy as np
import matplotlib.pylab as plt
```

CNN (합성곱 신경망)

국부수용장

- 수용장: 외부 자극이 특정 뉴런을 활성화 시키는 것
- 국부수용장: 시각 피질에서 국부적으로 수용을 한다?
- ▪ → 사물을 볼 때 하나가 아니라 여러 부분을 통해 인지한고, 각 수용장은 겹치지 때문에 그 부분을 모으면 시각의 전체를 의미한다
- ---신경망에서 이미지를 인식하는 것도 국부수용장에서 아이디어를 얻어서 개발하기 시작

합성곱 연산 해보기

■ 합성곱 연산

- ▪ 이미지 처리에서 말하는 필터 연산에 해당함

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

입력 데이터

2	0	1
0	1	2
1	0	2

필터

15	16
6	15

필터 배열

[139]:
```
b = np.array([
    [2, 0, 1],
    [0, 1, 2],
    [1, 0, 2]
])
```

1번 합성곱

1	2	3
0	1	2
3	0	1

[140]:
```
r1 = np.array([
    [1, 2, 3],
    [0, 1, 2],
    [3, 0, 1]
])
print(r1*b)
r1 = np.sum(r1*b)
print(r1)
```

```
[[2 0 3]
 [0 1 4]
 [3 0 2]]
15
```

2번 합성곱

2	3	0
1	2	3
0	1	2

[141]:
```
r2 = np.array([
    [2, 3, 0],
    [1, 2, 3],
    [0, 1, 2]
])
print(r2*b)
r2 = np.sum(r2*b)
print(r2)
```

```
[[4 0 0]
 [0 2 6]
 [0 0 4]]
16
```

3번 합성곱

0	1	2
3	0	1
2	3	0

```
[142]: r3 = np.array([
           [0, 1, 2],
           [3, 0, 1],
           [2, 3, 0]
       ])
       print(r3*b)
       r3 = np.sum(r3*b)
       print(r3)
```

```
[[0 0 2]
 [0 0 2]
 [2 0 0]]
6
```

4번 합성곱

1	2	3
0	1	2
3	0	1

```
[143]: r4 = np.array([
           [1, 2, 3],
           [0, 1, 2],
           [3, 0, 1]
       ])
       print(r4*b)
       r4 = np.sum(r4*b)
       print(r4)
```

```
[[2 0 3]
 [0 1 4]
 [3 0 2]]
15
```

결과

15	16
6	15

```
[144]: np.array([
           [r1, r2],
           [r3, r4]
       ])
```

```
[144]: array([[15, 16],
              [ 6, 15]])
```

출력 사이즈 구하기

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

```
[149]: def getOwOh(W, H, FH, FW, padding = 0, stride = 1):
           oh = (H + (2*padding) - FH) / stride + 1
           ow = (W + (2*padding) - FW) / stride + 1
           return (ow, oh)
```

- 예제1) 입력 크기 = (4, 4), 패딩 = 1, 스트라이드 = 1, 필터 = (3, 3) 일때 출력 크기는?

```
[150]: padding = 1
       stride = 1
       W, H = 4, 4
       FW, FH = 3, 3

       o = getOwOh(W, H, FH, FW, padding, stride)
       o
```

```
[150]: (4.0, 4.0)
```

- 예제2) 입력 크기 = (7, 7), 패딩 = 0, 스트라이드 = 2, 필터 = (3, 3) 일때 출력 크기는?

```
[151]: padding = 0
       stride = 2
       W, H = 7, 7
       FW, FH = 3, 3

       o = getOwOh(W, H, FH, FW, padding, stride)
       o
```

```
[151]: (3.0, 3.0)
```

- 예제3) 입력 크기 = (28, 31), 패딩 = 2, 스트라이드 = 3, 필터 = (5, 5) 일때 출력 크기는?

```
[152]: padding = 2
       stride = 3
       W, H = 28, 31
       FW, FH = 5, 5

       o = getOwOh(W, H, FH, FW, padding, stride)
       o
```

```
[152]: (10.0, 11.0)
```