

딥러닝시스템

세종대학교 소프트웨어융합대학 지능기전공학부

딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

Structuring DL projects and hyperparameter tuning

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

- 학습 목표

- 성능 지표 정의하기
- 베이스라인 모델 설정하기
- 학습 데이터 준비하기
- 모델을 평가하고 성능 개선하기

- 학습 내용

- 빠르고 효율적으로 동작하는 딥러닝 시스템을 만드는 방법과 결과를 분석하고 성능을 개선하는 방법을 살펴봄

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.1 성능 지표란

- 성능 지표를 통해 시스템을 평가할 수 있음
- 모델의 성능을 평가하는 가장 간단한 수단은 정확도
- 정확도는 모델의 예측이 정답과 일치한 비율로 정의
 - 예) 100개의 입력 표본을 대상으로 90개에 대해 모델이 정확한 예측을 내렸다면 이 모델의 정확도는 90%

$$\text{정확도} = \frac{\text{정답을 맞힌 횟수}}{\text{전체 표본 수}}$$

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.1.1 정확도가 가장 좋은 지표인가

▪ 문제 마다 적절한 평가 지표가 존재함

▪ 예) 희귀한 질환의 유무(1백만명 중 1명 꼴로 발병)를 판정하는 진단 모델 설계

- 99.999% 정확도를 갖는다고 표현하는게 옳을까?
- 정확도만 보면 매우 높은 성능처럼 보이지만 이 시스템으로는 실제 질환을 가진 사람을 찾아낼 수가 없어, 모델의 성능을 평가하기 적합한 지표가 아님

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

4.1.2 혼동행렬confusion matrix

- 혼동행렬은 모델의 분류 결과를 정리한 표
 - 모델로부터 질환이 없다고 음성판정을 받은 환자는 추가 검사 없이 귀가
 - 반면 모델로부터 질환이 있다고 양성 판정을 받은 환자는 추가 검사 실시

	질환이 있다고 예측(양성)	질환이 없다고 예측(음성)
질환이 있음(양성)	100 진양성(TP)	30 위음성(FN)
질환이 없음(음성)	70 위양성(FP)	800 진음성(TN)

True Positive(TP): 모델이 양성이라고 정확하게 예측(질환이 있음)

True Negative(TN): 모델이 음성이라고 정확하게 예측 (질환이 없음)

False Positive(FP): 실제로는 음성이지만 모델이 양성이라고 잘못 예측 (1종 오류)

False Negative(FN): 실제로는 양성이지만 모델이 음성이라고 잘못 예측 (2종 오류)

- 어떤 오류가 치명적인가?
 - 위양성결과보다 위음성 결과가 치명적임
 - 따라서 **재현율**을 통해 모델을 평가하는 것이 옳음

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.1.3 정밀도와 재현율

- **재현율** *recall* (또는 민감도 *sensitivity*)은 모델이 질환이 있는 사람을 얼마나 잘못 진단했는지 알려줌

- 질환이 있는 사람을 음성으로 진단한 위음성이 얼마나 되는지 나타내는 지표

$$\text{재현율} = \frac{\text{진양성}}{\text{진양성} + \text{위음성}}$$

- **정밀도** *Precision* (또는 특이성 *Specificity*)는 재현율의 반대 개념으로, 모델이 질환이 없는 사람을 얼마나 잘못 진단했는지 알려줌

- 질환이 없는 사람을 양성으로 진단한 위양성이 얼마나 되는지 나타내는 지표

$$\text{정밀도} = \frac{\text{진양성}}{\text{진양성} + \text{위양성}}$$

- 스팸 메일 분류기의 경우, 정밀도가 적합한 지표
 - 수신인이 받아야 할 중요한 메일을 잘못된 스팸 분류 탭에 전달받지 못하면 안되기 때문

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

4.1.4 F-점수

- 재현율(r)과 정밀도(p)를 단일 지표로 한꺼번에 나타내고 싶은 경우 사용
- F-점수는 정밀도와 재현율의 조화평균^{harmonic mean}으로 정의

$$F - \text{점수} = \frac{2pr}{p + r}$$

*위양성: 질환이 없는 사람을 질환이 있다고 판단

- 예) 질환 판정 모델은 재현율이 더 중요한 모델이지만, 위양성 건수가 많아 정밀도가 낮다면 불필요하게 추가 검사를 받는 환자가 늘어난다는 것을 의미함. 따라서 재현율이 중요하지만 정밀도 역시 함께 살펴봐야함

	정밀도	재현율	F-점수
분류기A	95%	90%	92.4%
분류기B	98%	85%	91%

⊙ 모델 평가 지표는 향후 시스템의 개선 방향을 결정하는 중요한 요소이므로, 지표를 명확하게 정의하지 않으면 머신러닝 모델 변경이 시스템의 성능 개선으로 이어질 지 분명하게 판단하기 어려움

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.2 베이스라인 모델 설정 하기

- 문제의 종류에 따라 신경망 구조에 맞는 베이스라인 모델을 설정
- 베이스라인 모델 설정 시 고려사항
 - 어떤 유형의 신경망을 사용해야 하는가? (MLP, CNN 등)
 - YOLO나 SSD 등의 물체 인식 기법을 적용해야 하는가?
 - 신경망의 층수는 얼마나 두어야 할까?
 - 활성화 함수는 어떤 것을 사용할까?
 - 최적화 알고리즘은 어떤 것을 사용할까?
 - 드롭아웃, 배치 정규화 등의 규제화 기법을 사용해서 과적합을 방지해야 하는가?
- 만일 해결하려는 문제의 연구가 상당히 진행되었다면, 기존 모델과 알고리즘을 답습하는 것이 좋음
- 모델을 처음부터 학습하지 않고 다른 데이터셋으로 이미 학습된 모델을 가져와서 사용하는 방법을 사용하는 것도 좋음 → 전이학습^{transfer learning}

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.3 학습 데이터 준비하기

- 데이터 준비 과정은 문제의 유형과 데이터의 성격에 따라 크게 달라짐
- 해당 절에서는 학습을 시작하기 전에 필요한 기본적인 데이터 준비 기법을 다룸

■ 4.3.1 훈련 데이터, 검증 데이터, 테스트 데이터로 분할하기

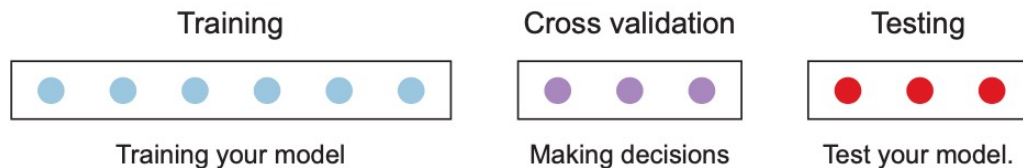
- 머신러닝 모델을 학습하기 위해 학습 데이터를 훈련 데이터와 테스트 데이터로 분할함
- 훈련데이터는 실제 학습에 사용하고, 테스트 데이터를 이용해서 학습된 모델의 성능을 평가
 - 절대, 테스트 데이터를 학습에 사용해서는 안됨
 - 학습 중에 테스트 데이터를 모델에 노출시키는 것은 시험에서 부정행위를 저지르는 것과 다를 바 없음



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

4.3.1.1 검증 데이터란

- 학습 중 한 에포크가 끝날 때마다 모델의 정확도와 오차를 확인해서 모델의 성능을 체크하고 하이퍼파라미터를 튜닝함
- 이 과정에서 테스트 데이터를 사용한다면 학습 중에 모델을 테스트 데이터에 노출시켜서는 안된다는 원칙이 깨짐
- 테스트 데이터는 학습이 완료된 후 최종 성능을 측정하는 목적으로만 사용
- 따라서, 훈련 데이터를 다시 분할한 별도의 데이터셋을 이용해서 학습 중 파라미터를 튜닝할때 사용하는데 이 데이터 셋을 **검증 데이터(validation data)**라고 함



각 에포크마다 모든 훈련 데이터에 대해 다음을 반복
신경망에 오차 역전파
가중치 수정
훈련 데이터를 대상으로 한 모델의 정확도와 오차 계산

모든 검증 데이터에 대해 다음을 반복
검증 데이터를 대상으로 한 모델의 정확도와 오차 계산
기존 모델과 비교해서 모델 갱신

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

- 4.3.1.2 훈련 데이터, 검증 데이터, 테스트 데이터를 잘 분할하는 방법
 - 전통적인 분할 방법 (데이터 셋의 규모가 수 만개에 지나지 않던 시절)
 - 훈련 데이터와 테스트 데이터의 비중은 80:20 혹은 70:30의 비율을 많이 사용
 - 검증데이터를 추가해야 한다면 60:20:20 또는 70:15:15 의 비율을 많이 사용
 - 최근 분할 방법 (데이터 셋의 규모가 큰 경우)
 - 테스트 데이터와 검증 데이터를 전체의 1%로 충분
 - 예) 데이터 수가 1백만개라면 테스트 데이터와 검증 데이터를 각각 1만 개씩 분할하면 충분
 - 데이터 분할 시 주의할 점
 - 같은 데이터 분포를 따르는지 확인할 것
 - 예) 훈련데이터는 인터넷에서 크롤링한 데이터(고품질 이미지) 사용, 테스트 데이터는 핸드폰으로 촬영한 데이터(저품질 이미지) 사용

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.3.2 데이터 전처리

- 전처리 기법은 다양하나 학습에 사용할 데이터셋의 상태나 문제의 유형에 따라 적절히 선택해야 함
- 신경망은 복잡한 데이터 전처리를 필요로 하지 않으며 충분한 양의 데이터가 제공되면 전통적인 머신러닝 기법과 달리 원 데이터를 그대로 입력해도 특징을 잘 학습함
- 그러나 모델의 성능을 개선하거나 신경망 구조의 한계로 인해 **기본적인 전처리**는 여전히 필요
 - 예) 컬러이미지 회색조로 변환, 이미지 크기 조절, 정규화, 데이터 강화 등

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.3.2.1 회색조 이미지 변환

- 컬러이미지는 3개의 행렬이 필요하여 회색조이미지보다 학습 계산 복잡도가 높음
- 문제 해결에 색상 정보가 불필요하거나 학습 계산 복잡도를 경감시켜야 하는 경우라면 컬러 이미지를 회색조 이미지로 변환하는 것을 검토해볼만 함

■ 4.3.2.2 이미지 크기 조절

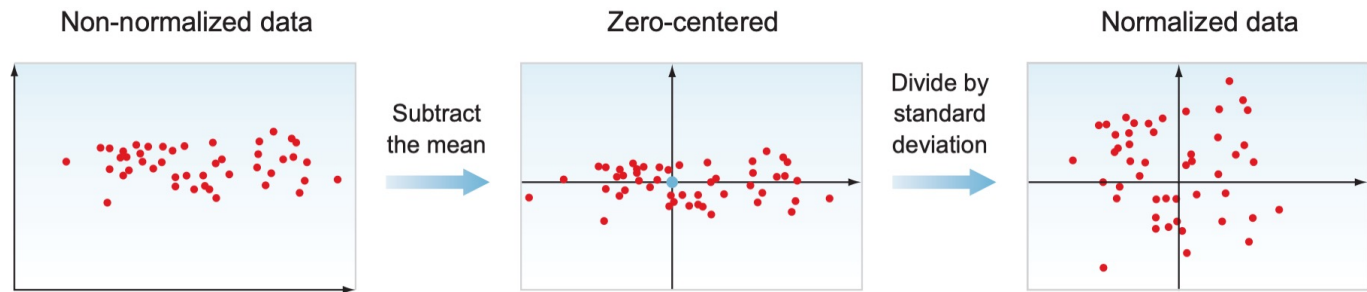
- 신경망의 한계점 중 하나는 입력되는 모든 이미지의 크기가 같아야 한다는 것
 - MLP를 사용하려면 입력층의 노드수가 이미지의 픽셀 수와 같아야 함
 - CNN 역시 첫번째 합성곱 층의 입력 크기를 이미지 크기에 맞춰 설정해야 함

```
torch.nn.Conv2d(in_channels=3, out_channels=32,  
                 kernel_size=3, stride=1, padding=1)
```

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.3.2.3 데이터 정규화

- 데이터 정규화란 데이터에 포함된 입력 특징 (이미지의 경우 픽셀값)의 배율을 조정해서 비슷한 분포를 갖게 하는 것
 - 정규화란 모든 픽셀값의 평균과 표준편차를 구한 다음 각 픽셀값에서 이 평균을 빼고 표준편차로 나누는 방법
 - 훈련 데이터와 테스트 데이터를 동일한 평균과 표준편차로 정규화해야 함

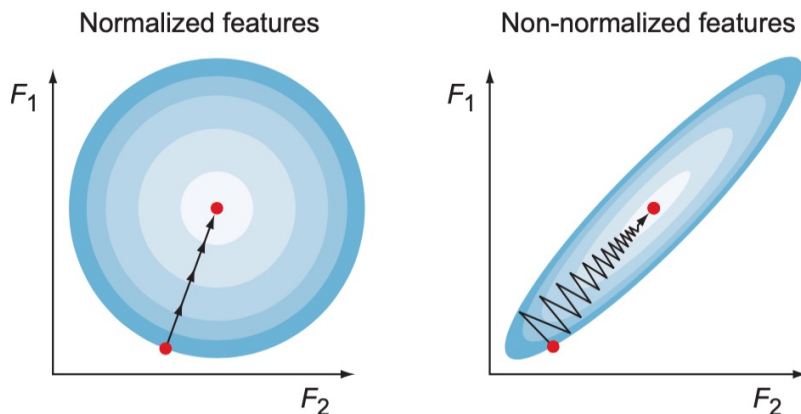


4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

4.3.2.3 데이터 정규화

- 원본 이미지는 서로 다른 배율(픽셀값의 범위)을 가진 픽셀로 구성되는 경우가 있음
 - 픽셀 값의 범위가 $[0, 255]$ 인 이미지가 있는가 하면, $[20, 200]$ 일 수도 있음
 - 정규화가 반드시 필요하지는 않지만, 정규화를 진행하면 학습된 모델의 성능이 개선되거나 학습 시간이 짧아지는 장점이 있음

정규화 유무에 따른 경사하강법 학습 과정



F_1, F_2 는 입력데이터의 특징

특징을 정규화하면 전역최소점에 빠르게 도달
정규화하지 않으면 가장 경사가 가파른 방향이
계속 진동하므로 전역 최소점에 도달이 늦어짐

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.3.2.4 데이터 강화

- 데이터 강화(Data Augmentation)는 규제화를 설명할 때 자세히 다룸
- 하지만 데이터 강화를 데이터 전처리로도 활용할 수 있음

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

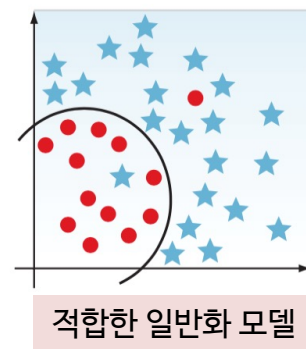
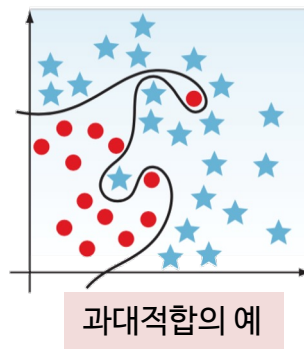
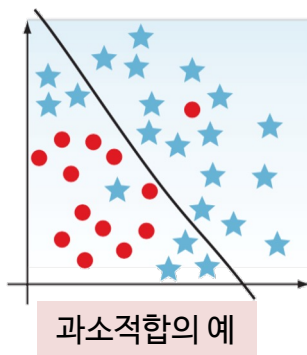
- 4.4 모델을 평가하고 성능 지표 해석하기

- 베이스라인 모델 설정 → 데이터 전처리 → 모델 학습 → 모델 평가 → 성능 지표 해석
- 성능지표해석
 - 모델 학습이 끝나면 모델이 제 성능을 내지 못하는 구성 요소가 없었는지 살펴보고 성능이 만족스럽지 못하면 그 원인이 무엇인지(과적합, 과소적합, 데이터 결함 등)확인 하는 과정

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.4.1 과적합의 징후

- 과소적합은 모델의 표현력이 데이터에 비해 부족해서 발생하는 현상
 - 과소적합의 대표적인 예: 단일 퍼셉트론을 사용해서 아래의 데이터 분류 불가
- 과대적합은 반대로 모델이 지나치게 복잡한 경우 발생하는 현상
 - 훈련 데이터의 특징을 학습하는 대신 훈련 데이터 자체를 기억하는 현상
 - 훈련 데이터를 대상으로는 높은 성능을 보이지만 테스트 데이터처럼 처음 접하는 데이터를 대상으로 하는 **일반화 성능이 낮음**
 - 과적합의 대표적인 예: 모델이 훈련 데이터를 지나치게 배우
- 바람직한 모델은 데이터에 적합한 복잡도를 가짐



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

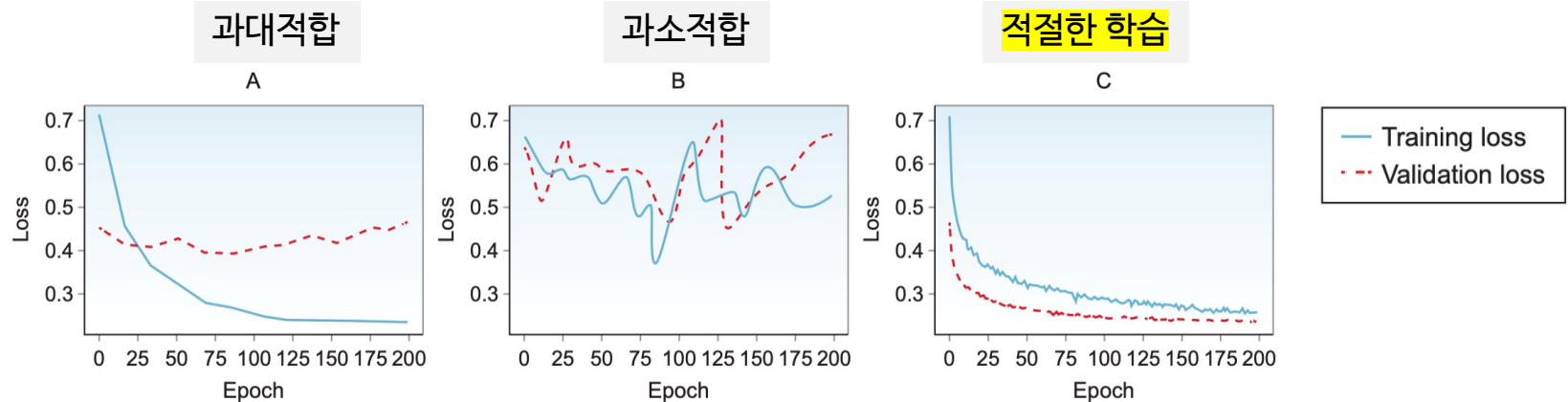
▪ 4.4.1 과적합의 징후

- 훈련 데이터에 대한 성능은 높으며 검증 데이터에 대한 성능이 상대적으로 낮다면 **과대적합**을 일으키고 있을 가능성이 높음
 - 예를 들어, train_error가 1%, val_error가 10% 라면 모델이 훈련 데이터 자체를 기억했기 때문에 검증 데이터에서 성능이 제대로 나오지 않는다는 뜻
 - 시행착오를 거치며 적절한 성능이 나올 때까지 **하이퍼파라미터를 조정**해야 함
- 훈련 데이터에 대한 성능이 낮으면 **과소적합**을 일으키고 있을 가능성이 높음
 - 예를 들어, train_error가 14%고 val_error가 15%면 모델의 표현력이 낮아 데이터에 부합하지 못하는 것
 - 신경망에 은닉층을 추가하거나 학습 에포크 수를 늘리거나 다른 신경망 구조를 사용해야 함

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

4.4.2 학습 곡선 그리기

- 학습 오차와 검증 오차의 추이를 살펴볼 때는 그래프를 통해 확인하는 것이 좋음



과대적합의 징후:

훈련데이터에 대한 손실값이 개선되고 있지만 검증 데이터에 대한 일반화 성능이 나오지 않고 있음

과소적합의 징후:

훈련데이터와 검증데이터 모두에서 성능이 나오지 않는 신경망이며, 실질적으로 학습이 제대로 되지 않는 상황임. 신경망이 너무 간단하여 이미 가지고 있는 데이터에서 배울 수 없기 때문에 데이터가 더 필요하지 않음

적절한 학습의 징후:

훈련 데이터와 검증 데이터 모두에서 손실 값이 개선되고 있음

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.4.3 실습: 신경망의 구성, 학습, 평가

- 하이퍼파라미터 튜닝에 들어가기 전에 데이터 분할과 모델 구성, 학습, 성능 측정 결과의 시각화 과정을 빠르게 실습해보자

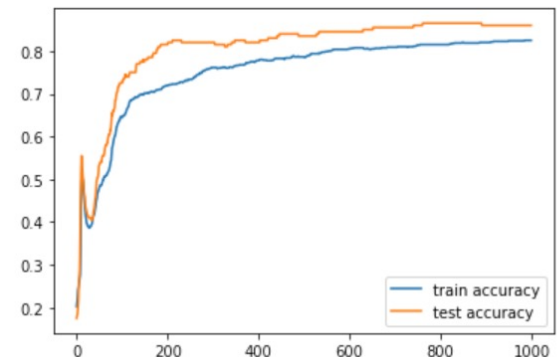
■ 실습 내용

- 80:20의 비율로 훈련 데이터와 테스트 데이터 분할
- MLP 모델 구성
- 모델 학습
- 모델의 성능 측정
- 성능 측정 결과 시각화

■ 실습 코드

- <https://www.kaggle.com/code/sukzoon1234/2024-1-dls-w4>

Error 그림을 그려보는 것도 좋음



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

- 4.5 신경망을 개선하고 하이퍼파라미터 튜닝하기
- 4.5.1 데이터 추가 수집 또는 하이퍼파라미터 튜닝
 - 판단 기준 사례
 - 훈련 데이터에 대한 기존 성능이 납득할만한 수준인지 확인
 - 훈련 데이터 정확도와 검증 데이터 손실 값 혹은 정확도를 시각화해서 관찰
 - <과소적합 징후>가 보인다면 데이터의 추가 수집 없이 하이퍼파라미터를 조정하거나 기존 훈련 데이터를 전처리 하는 것이 좋음
 - <과대적합 징후>가 보인다면 데이터 추가 수집이 유효한 경우에 해당됨

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.5.2 파라미터와 하이퍼파라미터

- 하이퍼파라미터: 우리가 값을 정하고 조정할 수 있으며 신경망의 학습 대상이 아님
 - 예) 학습률, 배치 크기, 에포크 수, 은닉층 수
- 파라미터: 학습 과정을 통해 신경망이 조정하며 우리가 직접 값을 수정하지 않음
 - 예) 가중치와 편향

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.5.3 신경망의 하이퍼파라미터

- 하이퍼파라미터 튜닝의 어려운 점: 어떤 데이터셋으로 어떤 문제를 해결하느냐에 따라 달라지기때문에, 모든 상황에 유효한 값이 없음
- 신경망의 하이퍼파라미터
 - 신경망 구조 (3번)
 - 은닉층 수(신경망의 깊이) / 각층의 뉴런 수 (층의 폭) / 활성화 함수의 종류
 - 학습 및 최적화 (1번)
 - 학습률과 학습률 감쇠 유형 / 미니배치 크기 / 최적화 알고리즘 종류 / 에포크 수 (조기 종료 적용 여부 포함)
 - 규제화 및 과적합 방지 기법 (2번)
 - L2 규제화 / 드롭아웃 / 데이터 강화

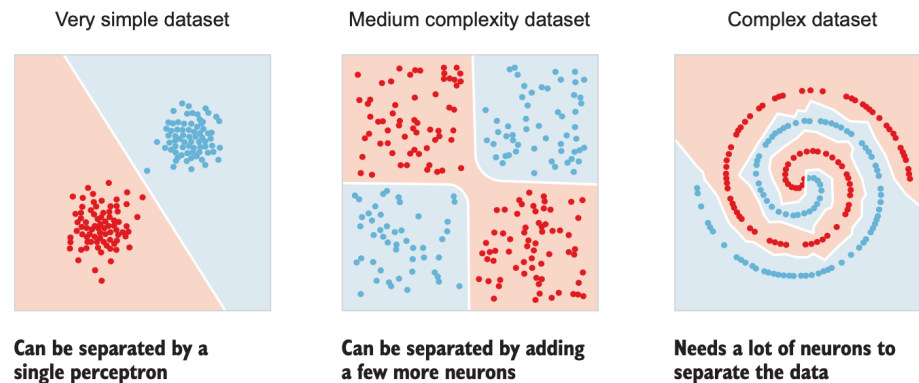
4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.5.4 신경망 구조

- 은닉층 수(신경망의 깊이) / 각층의 뉴런 수 (층의 폭) / 활성화 함수의 종류

■ 4.5.4.1. 신경망의 깊이와 폭

- 신경망의 깊이와 폭은 신경망의 학습 능력과 직결됨
- 신경망의 규모가 작으면 과소적합이 발생, 규모가 너무 크면 과대적합 발생
- 신경망의 적당한 규모를 가늠하려면 시작점을 선택하고 성능을 관찰한 다음 규모를 가감(+ 혹은 -)해야 함
- 데이터 셋이 복잡할 수록 신경망의 학습 능력이 많이 필요함



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.5.4.1. 신경망의 깊이와 폭

- 모델의 학습 능력이 지나치게 높으면 (은닉층 수 또는 유닛 수가 과도하게 많은 경우) 과적합을 일으켜 훈련 데이터 자체를 기억하는 현상이 발생 → 은닉층의 유닛수 감소
- 모델의 학습 능력이 지나치게 높아도 적절한 규제화(드롭아웃 또는 기타)를 적용하면 과적합으로 인한 성능 저하가 발생하지 않음
- 모델의 학습 능력이 부족하면 (은닉층 수 또는 유닛 수가 부족한 경우) 과소적합을 일으킴 → 은닉층의 유닛수 증가

■ 4.5.4.2. 활성화 함수의 종류

- 활성화 함수는 뉴런에 비선형성을 도입하는 수단
- 활성화 함수는 연구가 활발한 분야
- 일반적으로 ReLU와 그 변종의 성능이 우수하다고 알려져 있음

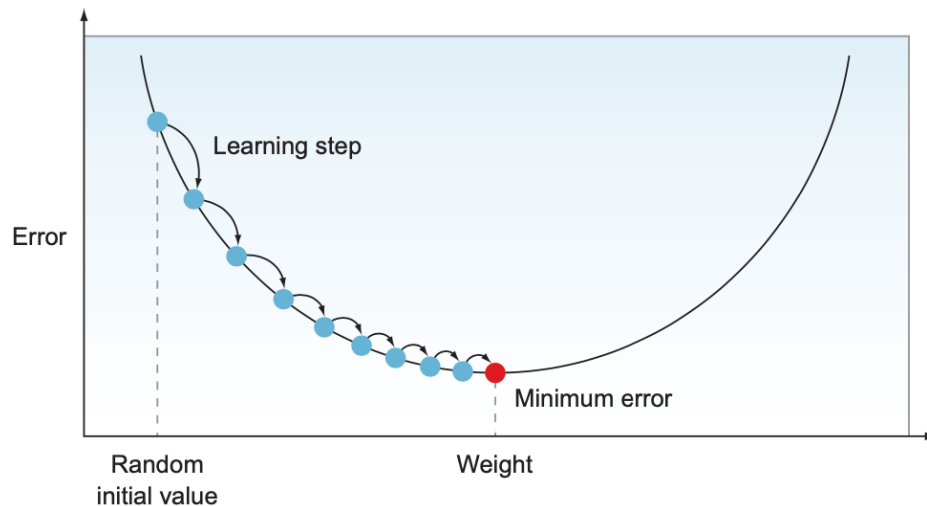
4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.6 학습 및 최적화

- 신경망 구조를 확정했으면, 학습과 최적화 과정에 관여하는 하이퍼파라미터를 살펴보자.

■ 4.6.1 학습률과 학습률 감쇠 유형

- 최적화 알고리즘 설정에서 오차 함수의 경사를 어느 정도의 보폭으로 내려갈지 결정해야 하는데 이 **보폭이 바로 학습률**을 의미함
- 학습률은 오차 함수의 경사를 얼마나 빨리 내려갈지를 결정함



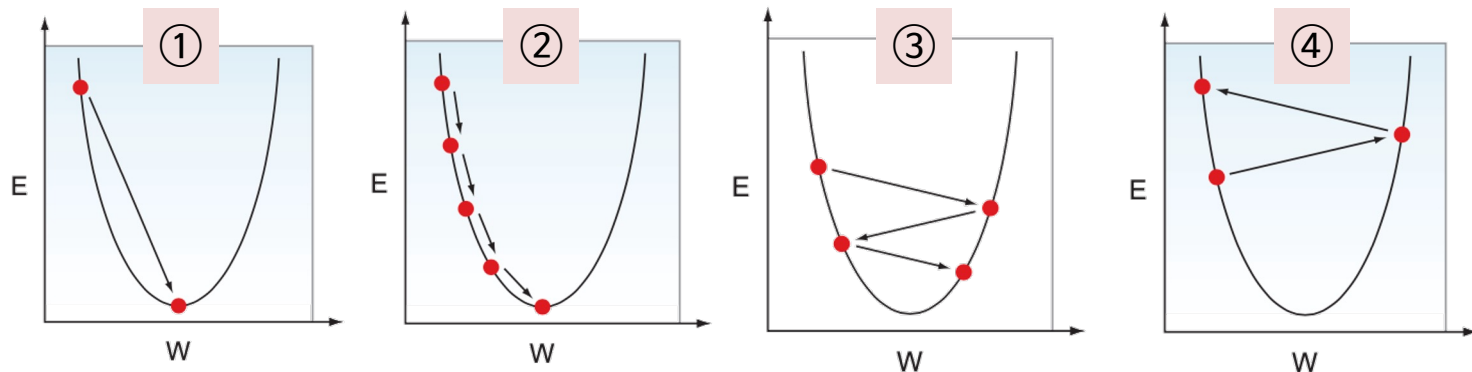
학습률은 가장 중요한 하이퍼파라미터로, 항상 잘 조정되어 있어야 한다. 단 하나의 하이퍼파라미터를 조정할 시간 밖에 주어지지 않는다면 학습률을 조정해야 한다.

-요수아 벤지오-

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

4.6.1 학습률과 학습률 감쇠 유형

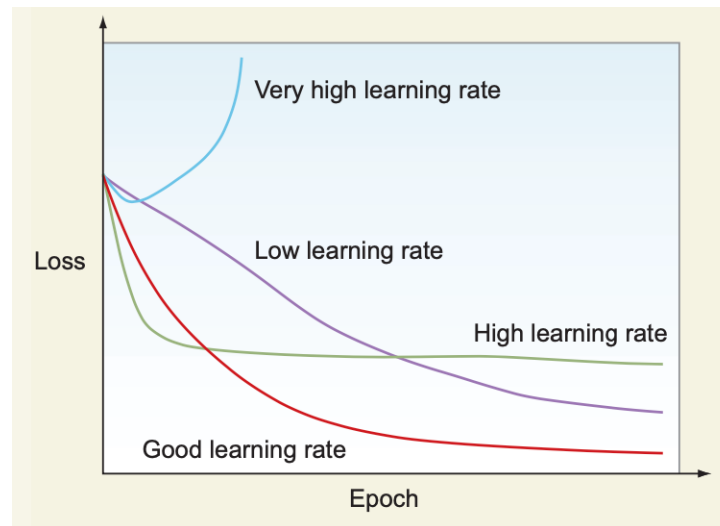
- ① 학습률을 절묘하게 이상적으로 설정한다면, 한번에 최소점에 도달
- ② 학습률이 이상적 학습률보다 작다면 또는 훨씬 작다면, 시간이 걸릴 뿐 결국 최소점에 도달
- ③ 학습률이 이상적 학습률보다 크다면, 그럭저럭 괜찮은 지점에서 파라미터가 수렴할 수 있지만 우리가 원하는 최소점과는 거리가 있음
- ④ 학습률이 이상적 학습률보다 훨씬 크다면, 가중치는 최소점을 지나쳐 오히려 원래보다 더 최소점에서 멀어지게 되며, 이러한 현상을 발산이라고 함



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.6.1 학습률과 학습률 감쇠 유형

- 학습률의 값은 최적화 속도와 성능의 트레이드 오프 관계
- 에포크 수에 따른 손실 값의 추이를 그래프로 나타내면 아래의 사실 관찰 가능
 - 작은 학습률: 손실은 계속 감소하지만 수렴까지 시간이 훨씬 오래 걸림
 - 큰 학습률: 학습 전과 비교하면 손실은 작지만 최소점과는 거리가 멀
 - 아주 큰 학습률: 초기에는 손실이 감소하지만 어느 순간 손실이 증가
 - 적당한 학습률: 손실이 일정하게 감소하며 최솟값에 도달



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

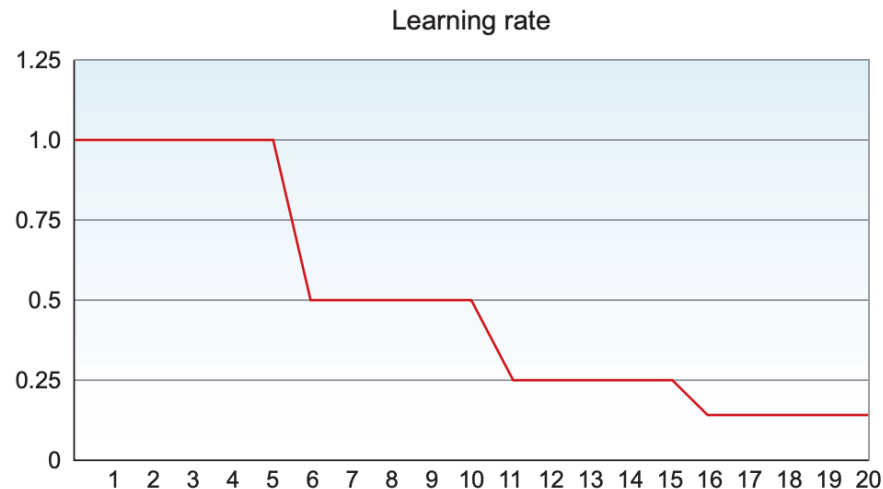
▪ 4.6.2 최적의 학습률을 정하는 방법

- 모든 딥러닝 라이브러리에서 **기본 값으로 미리 설정된 학습률은 좋은 출발점** 역할을 함
 - 각각의 최적화 알고리즘에도 미리 설정된 기본값이 있음
- 학습 시 출력되는 검증 데이터에 대한 손실 값을 분석하는 법
 - 파라미터가 수정될 때마다 `val_loss` 가 감소한다면 설정된 하이퍼파라미터는 정상임.
개선이 멈출 때까지 학습을 계속 진행
 - 학습이 끝나고 `val_loss` 가 계속 감소 중이라면, 학습률이 너무 작아 파라미터가 수렴하지 못한 상태. 이런 경우에는 다음 두가지 방법을 사용
 - 학습률은 그대로 두되 에포크 수를 늘려 학습을 다시 시작
 - 학습률을 조금 증가시키고 학습을 다시 시작
 - `Val_loss` 가 증감을 반복하며 진동한다면 학습률이 너무 큰 것
 - 학습률을 감소 시켜 학습을 다시 시작

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.6.3 학습률 감쇠와 적응형 학습

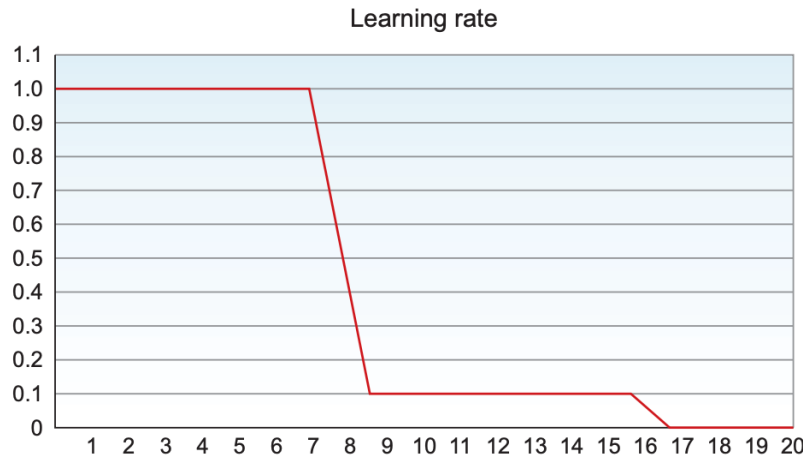
- 학습률 감쇠^{learning rate decay} 는 학습을 진행하는 도중 학습률을 변화시키는 방법
- 대부분 고정된 학습률보다 성능이 뛰어나며 학습 시간을 크게 줄이는 효과도 있음
- 학습 초기에는 큰 학습률을 적용하다가 점차 학습률을 감소시켜 최소점을 지나치는 현상을 방지
- 학습률 감쇠에도 여러가지 종류가 있음
 - 〈계단형 감쇠〉는 일정 비율로 학습률을 감소시킴



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.6.3 학습률 감쇠와 적응형 학습

- 〈지수형 감쇠〉는 8 에포크마다 학습률에 0.1을 곱하는 지수 감쇠를 적용
 - 계단형 감쇠에 비하면 파라미터가 수렴하기까지 오랜 시간이 걸리지만 수렴에 다다를 수 있음

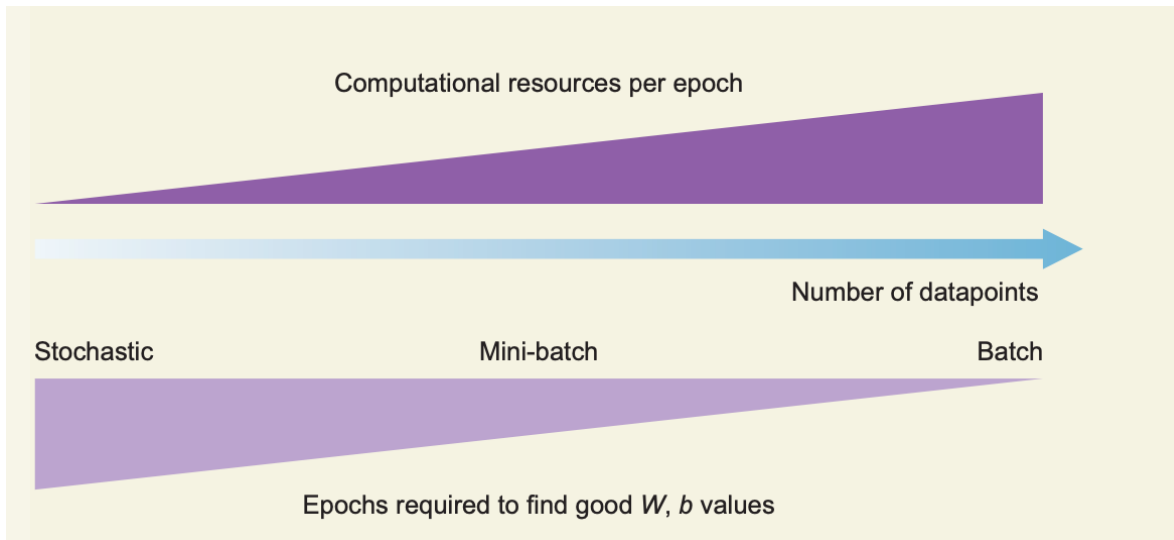


- 〈적응형 학습〉은 학습의 진행이 멈추는 시점에 학습률을 경험적으로 설정된 값만큼 자동으로 수정하는 방식
 - 학습률이 필요한 시점에 감소만 하는 것이 아니라 학습 속도가 지나치게 느려지는 상황에 따라서 증가하기도 함
 - 적응형 학습은 일반적인 학습률 감쇠 기법보다 높은 성능을 보임
 - Adam과 Adagrad가 적응형 학습이 적용된 최적화 알고리즘임

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.6.4 미니배치 크기

- Batch_size는 필요한 리소스 요구 사항과 학습 속도에 큰 영향을 미침
- 사용 중인 데이터 셋의 크기가 작다면, BGD를 사용해도 빠른 시간안에 학습 가능
- 큰 데이터 셋을 사용 중이라면, 64 혹은 128로 시작하는 것이 좋으며, 만족스러운 학습 속도가 나올 때까지 32, 64, 128, 256, 512, 1024와 같이 배치 크기를 두배씩 늘림
- 그러나 이때 사용중인 컴퓨터의 메모리의 용량을 고려해야 함
- 1024 이상의 미니배치는 학습은 가능하지만 그리 널리 사용되지 않음



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.6.4 미니배치 크기

▪ 경사 하강법 리뷰

▪ 배치경사하강법 batch gradient descent, BGD

- 데이터 셋 전체를 한번에 신경망에 입력해서 순방향 계산, 오차 계산, 경사 계산을 거쳐 오차를 역전파시켜 가중치를 수정
- 최적화 알고리즘이 전체 훈련 데이터를 대상으로 오차를 계산하므로 가중치 수정은 한 에포크에 한번만 일어남
- 미니배치의 크기가 전체 데이터셋인 확률적 경사 하강법이라고 생각하면 됨
- BGD의 장점은 노이즈가 적고 최소점까지 큰 보폭으로 접근할 수 있다는 것
- 단점은 가중치를 한번 수정하는데 전체 데이터셋이 필요하므로 학습 속도가 느리며 데이터셋의 규모가 클수록 더 심함
- 또한 메모리 요구량도 증가하 BGD를 적용하지 못할 수도 있음
- 소규모 데이터셋을 사용할 때 유리

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.6.4 미니배치 크기

▪ 경사 하강법 리뷰

▪ 확률적경사하강법 stochastic gradient descent, SGD

- 온라인 학습 이라고도 함
- **훈련 데이터를 한번에 하나씩 신경망에 입력해서 순방향 계산, 오차, 경사 계산을 고쳐 오차를 역전파시켜 가중치를 수정함**
- SGD는 데이터 하나마다 가중치가 수정됨
- 따라서 SGD는 가중치의 진행 방향에 진동이 심하며, 때로는 엉뚱한 방향으로 나아가기도 함
- 이러한 노이즈는 학습률을 감소시켜 억제할 수 있는데 대체로 BGD보다 나은 성능을 보임SGD를 사용하면 전역 최소점에 빠르고 더 가까이 접근할 수 있음
- 반면 단점은 한번에 데이터 하나만 처리하므로 데이터 여러 개를 한번의 행렬 연산으로 처리하는 학습 계산 속도의 이점을 살리기 어려움

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.6.4 미니배치 크기

▪ 경사 하강법 리뷰

- 미니배치경사하강법 mini-batch gradient descent, MB-GD
 - 배치 경사 하강법과 확률적 경사 하강법의 중간 정도
 - 가중치를 한번 수정하는데 데이터셋 전체나 데이터 하나를 사용하는 대신 훈련 데이터를 몇개의 미니배치로 분할해서 사용
 - 이 방법을 통해 행렬곱을 이용한 계산 속도가 향상되고, 전체 데이터셋을 사용하지 않으므로 가중치를 한번 수정하는데 걸리는 시간 역시 짧아짐

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

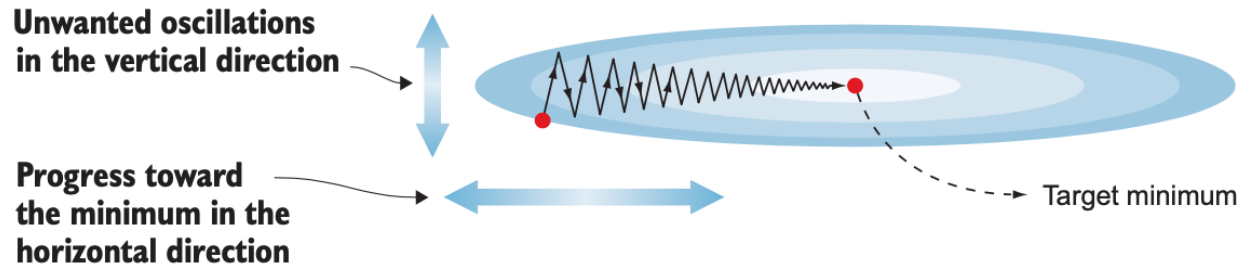
▪ 4.7 최적화 알고리즘

- 오랫동안 다양한 최적화 알고리즘이 제안되었으나, 이들은 특정 유형의 문제에서만 효과적임을 증명했음. 즉, 광범위한 문제에 일반적으로 적용할 수 없음
- 딥러닝 커뮤니티는 <경사 하강법>과 <그 변종>이 효과적이라는 사실을 알게 됨
- 경사하강법은 학습률을 너무 작게 설정하면 학습이 오래 걸리고, 너무 크게 설정하면 손실이 감소하지 않거나 발산하여 학습이 되지 않는 현상이 발생함

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

4.7.1 모멘텀을 적용한 경사 하강법

- 확률적 경사 하강법은 오차의 최소점으로 향하면서 이동 방향에 진동이 일어남
- **가중치 이동 방향의 이러한 진동을 감소시키기 위해 모멘텀(momentum)이 고안됨**
- 모멘텀은 엉뚱한 방향으로 가중치의 이동 방향이 진동하는 것을 완화시키는 기법
- 모멘텀 기법은 경사가 기존 이동 방향과 같으면 이동 폭을 증가시키고, 기존 이동 방향과 다른 방향의 경사에는 이동 폭을 감소시킴
- 이런 방법을 통해 가중치의 진동을 완화시키고 더욱 빨리 수렴에 이를 수 있음



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

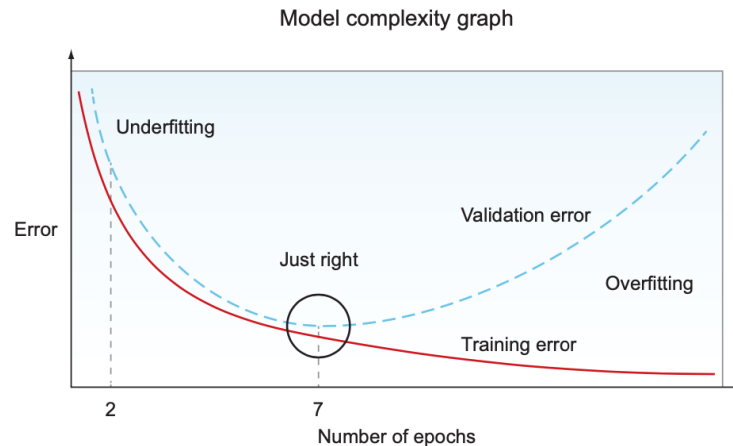
▪ 4.7.2 Adam

- Adam은 적응형 모멘트 예측^{adaptive moment estimation}의 약자
- Adam은 모멘텀과 비슷하게 이전에 계산했던 경사의 평균을 속도향으로 사용하지만 속도향이 지수적으로 감소된다는 차이가 있음
 - 모멘텀이 경사를 굴러 내려가는 공과 같다면, Adam은 무거운 공이 마찰력을 가진 바닥을 굴러 내려가며 모멘텀이 감소하는 것에 비유할 수 있음
- Adam은 다른 최적화 알고리즘보다 학습 시간이 빠르다는 장점이 있음
- Adam의 하이퍼파라미터는 학습률만 조정하면 됨

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.7.3 에포크 수와 조기 종료 조건

- 에포크^{epoch}는 학습 진행 중 전체 훈련 데이터가 한번 모델에 노출된 횟수를 의미함
- **에포크는 신경망의 반복 학습 횟수를 의미함**
- 반복 학습 횟수가 많을 수록 신경망은 더 많은 특징을 학습할 수 있음
- 신경망의 반복 학습 횟수가 충분한지 확인하려면 학습 중 훈련 데이터의 오차와 검증 데이터의 오차를 잘 관찰해야 함
- 훈련 오차와 검증 오차가 함께 개선되다가 검증 오차가 증가하면서 과적합 징후 발생
- 과적합이 일어나기 전에 학습을 종료하는 기법이 필요하며 이를 조기 종료라고 함



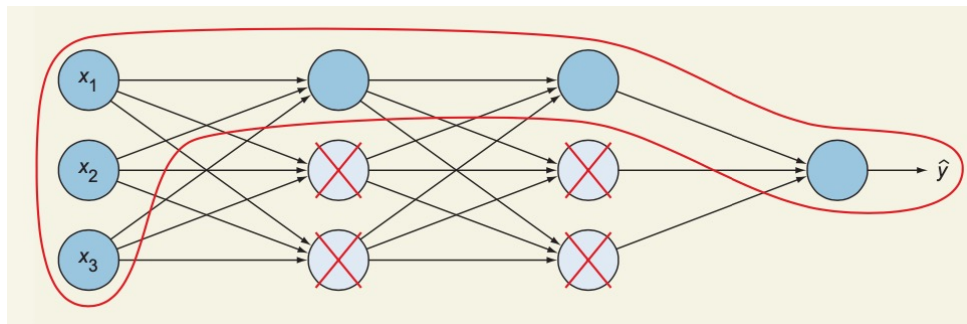
4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.8 과대적합을 방지하기 위한 규제화 기법

- 학습 중인 신경망에 과대적합이 발생했다면 신경망의 표현력을 감소시켜야 하며, 가장 먼저 시도할 수 있는 방법은 **규제화**임
- **규제화 기법으로는 L2 규제화, 드롭아웃, 데이터 강화가 있음**

■ 4.8.1 L2 규제화

- 오차 함수에 규제화항 regularization term 을 추가하는 것
- 은닉층 유닛의 가중치가 0에 가까워지고 모델의 표현력을 감소시키는데 도움이 됨
 - 가중치가 0에 가까워지면 뉴런의 역할이 상쇄되는 것



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.8.1 L2 규제화


▪ 규제화항 정의

- 람다는 규제화 파라미터, m 은 인스턴스 수, w 는 가중치

$$\text{error function}_{\text{new}} = \text{error function}_{\text{old}} + \text{regularization term}$$

$$\text{L2 regularization term} = \frac{\lambda}{2m} \times \sum \|w\|^2$$

$$\text{error function}_{\text{new}} = \text{error function}_{\text{old}} + \frac{\lambda}{2m} \times \sum \|w\|^2$$


$$W_{\text{new}} = W_{\text{old}} - \alpha \left(\frac{\partial \text{Error}}{\partial W_x} \right)$$

- L2규제화는 가중치를 0을 향해 감소시키기 때문에 가중치 감소 weight decay 라고 부르기도 함

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.8.1 L2 규제화

```
optimizier = torch.optim.Adam(model.parameters(),  
                                lr=1e-3, weight_decay=0.9)
```



- Lambda 값은 하이퍼파라미터로 직접 조정해야 하지만 기본값으로도 잘 작동함
- L2 규제화를 적용해도 과적합이 해소되지 않는다면, Lambda 값을 증가시켜 모델의 복잡도를 더욱 낮추면 됨

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

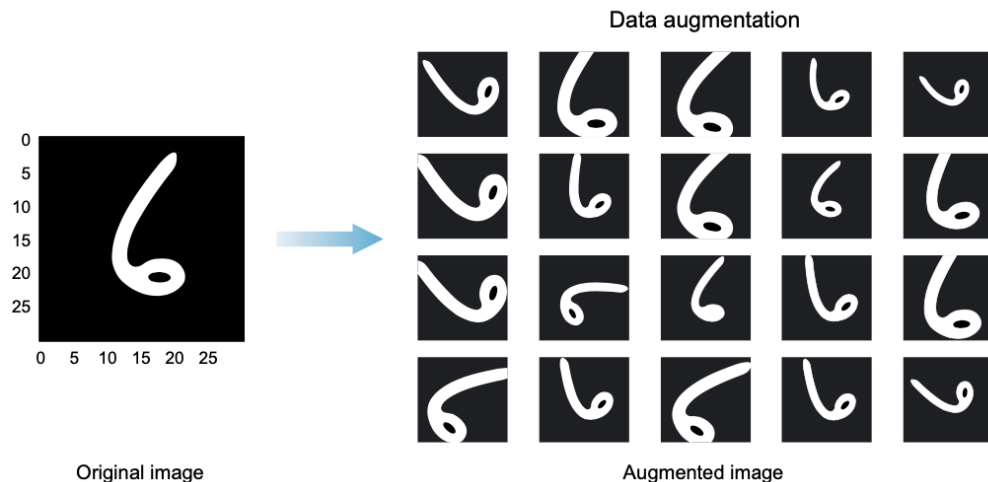
▪ 4.8.2 드롭아웃층

- 드롭아웃 역시 신경망의 복잡도를 낮춰 과적합을 방지하는 효과적인 방법
- 드롭아웃은 학습 반복마다 전체 뉴런 중 미리 정해진 비율 $p^{\text{dropout rate}}$ 만큼의 뉴런을 해당 반복 회차 동안 비활성화(드롭아웃) 하는 것
- P 는 0.3에서 0.5 사이의 값으로 설정함. 초기에는 0.3으로 시작해서 과적합이 발생하면 비율을 올려 대응하면 됨
- L2규제화와 드롭아웃 모두 뉴런의 효율을 떨어뜨려 신경망의 복잡도를 감소시킴
- 차이점은 L2규제화는 가중치를 통해 뉴런의 영향력을 억제한다면, 드롭아웃은 특정 뉴런의 영향력을 완전히 비활성화 함

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.8.3 데이터 강화

- 과적합을 방지하는 방법 중 하나는 학습 데이터를 추가하는 것
- 데이터 추가는 상황에 따라 가능하지 않은 경우도 있으나 기존 데이터에 약간의 변형을 가해 새로운 데이터를 만드는 것은 가능
- 데이터 강화는 저렴한 비용으로 훈련 데이터의 양을 늘려 과적합을 방지할 수 있는 기법
- 데이터 강화의 예로는 이미지 반전, 회전, 배율 조정, 밝기 조절 등 이 있음



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.8.3 데이터 강화

- 데이터 강화는 모델이 특징 학습 중 대상의 원래 모습에 대한 의존도를 낮춰준다는 의미에서 일종의 규제화 기법으로 취급되기도 함
- 데이터 강화는 신경망이 새로운 데이터에 노출되어도 유연하게 대응할 수 있는 능력을 길러줌

```
import torch
from torchvision.transforms import v2

H, W = 32, 32
img = torch.randint(0, 256, size=(3, H, W), dtype=torch.uint8)

transforms = v2.Compose([
    v2.RandomResizedCrop(size=(224, 224), antialias=True),
    v2.RandomHorizontalFlip(p=0.5),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
img = transforms(img)
```

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

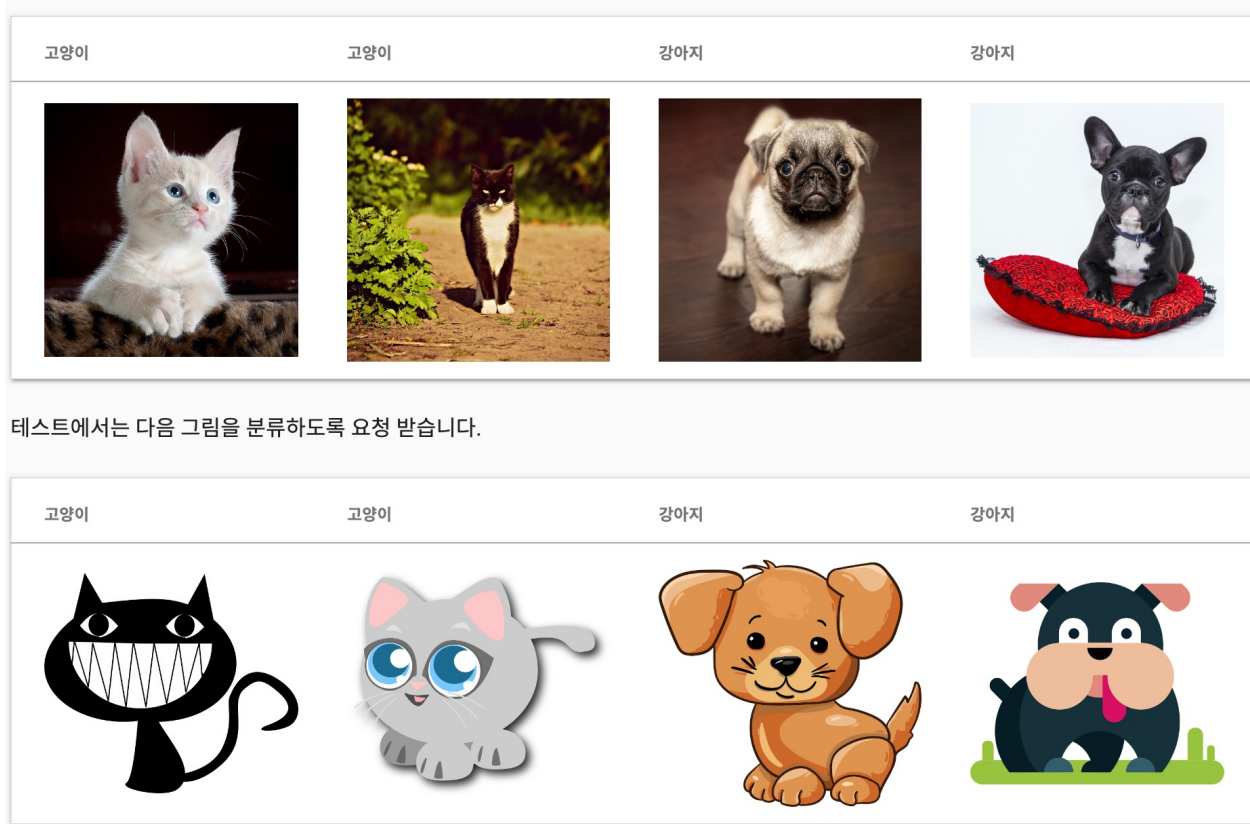
▪ 4.9 배치 정규화

- 앞부분에서 학습 속도를 개선하기 위한 **데이터 정규화**를 설명함
- 더불어 앞서 배운 정규화 기법은 입력층에 이미지를 입력하기 위한 학습 데이터의 전처리에 집중되어 있었음
- 추출된 특징을 정규화하면 은닉층도 정규화의 도움을 받을 수 있음
- 추출된 특징은 변화가 심하므로 정규화를 통해 신경망의 학습 속도와 유연성을 더욱 개선할 수 있음
- 이런 기법을 **배치 정규화** batch normalization, BN 이라고 함

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

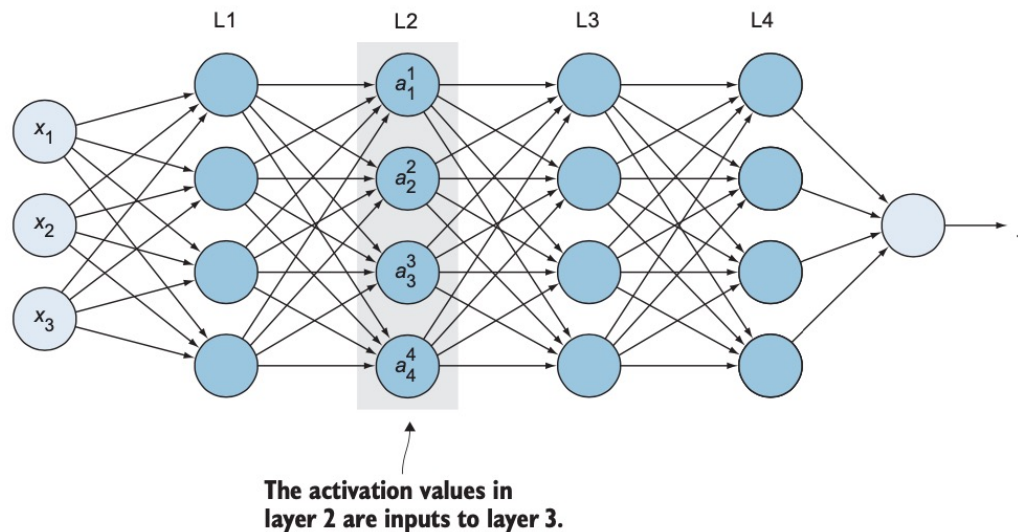
4.9.1 공변량 시프트 문제 (covariate shift)

- 공변량(학습 데이터)의 분포가 테스트 데이터의 분포가 다른 상황을 의미
- 공변량 시프트가 발생하면 모델을 다시 학습해야 할 수도 있음



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

- 4.9.2 신경망에서 발생하는 공변량 시프트 문제
 - 배치 정규화는 은닉층 출력의 분포가 변화하는 것을 억제 함
 - 즉, 값 자체는 변화하지만 값의 **평균과 분산은 변화하지 않음**



4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.9.3 배치 정규화의 원리

- 공변량 시프트를 완화하기 위한 대책으로 배치 정규화가 제안되었음
- 배치 정규화는 각 층의 활성화 함수 앞에서 다음 연산을 추가하는 방법임

- 입력의 평균을 0으로 조정 (입력미니배치의 평균과 표준편차를 계산)

$$\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad \leftarrow \text{Mini-batch mean}$$
$$\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 \quad \leftarrow \text{Mini-batch variance}$$

- 평균이 0으로 조정된 입력을 정규화 (ϵ 은 일반적으로 10^{-5})

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- 연산 결과의 배율(γ) 및 위치(β) 조정 (두 가지 파라미터를 찾기까지 약간 학습 속도가 느리지만 찾으면 빨라짐)

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

- 배치 정규화 기법을 적용하면 각 층의 입력이 최적의 배율과 평균으로 조정되어 학습을 진행할 수 있음

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.9.4 배치 정규화 구현하기

- 정규화된 결과를 다음 층에 전달할 수 있도록 은닉층 뒤에 배치 정규화층을 추가하는 형태로 구현됨

신경망에 배치 정규화를 추가한 예

```
import torch
import torch.nn as nn
import torch.nn.functional as F

model = nn.Sequential(
    nn.Linear(in_dim, mid_dim), # linear layer 추가
    nn.ReLU(), # 활성화함수 layer 추가
    nn.BatchNorm1d(mid_dim), # Batch norm layer 추가
    nn.Dropout(p=0.5), # Dropout layer 추가

    nn.Linear(mid_dim, out_dim), # linear layer 추가
    nn.ReLU(), # 활성화함수 layer 추가
    nn.BatchNorm1d(out_dim), # Batch norm layer 추가

    nn.softmax(dim=1) # 최종 softmax layer 추가
)
```

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

▪ 4.9.5 배치 정규화 복습

- 배치 정규화는 입력층뿐만 아니라 은닉층도 정규화의 장점을 누릴 수 있다는 점에서 출발
- 배치 정규화 덕분에 뒷층의 학습이 앞층의 학습 결과의 영향을 덜 받으므로 각 층마다 독립적인 학습이 가능하게 되었음
- 앞층의 출력이 항상 같은 평균과 분산을 갖게 되므로 뒷층의 관점에서는 입력이 크게 흔들리지 않는 효과가 있으며, 그만큼 뒷층의 학습이 쉬워지게 되었음
- 즉, 배치 정규화는 은닉층 유닛의 출력이 항상 표준 분포를 따르도록 강제하는 방법

4. 딥러닝 프로젝트 시동 걸기와 하이퍼파라미터 튜닝

■ 4.10 프로젝트: 이미지 분류 정확도 개선하기

이미지 분류 정확도 개선하기

10가지 클래스를 가진 데이터 셋 (CIFAR10)에 대한 이미지 분류를 수행합니다.

베이스라인 정보

- GPU: T4×2
- 모델
 - resnet50 사용
 - IMAGENET1K_V1 으로 가중치 초기화

```
epochs = 3
batch_size = 128
lr=1e-3
optimizer: SGD
```

데이터 증강 기법

- 우선 255로 나눠서 0~1 범위로 만들기
- 아래 Data Augmentation 기법을 사용하였으며, pytorch 공식 documentation을 참고하시면 됩니다.
 - <https://pytorch.org/vision/stable/transforms.html>
- <학습>
 - i) Resize(224×224) & Random Crop
 - ii) Horizontal Flip (p=0.5)
 - iii) Dtype torch.float32로 변경
 - iv) mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225]
- <평가>
 - i) Resize(224×224)
 - ii) Dtype torch.float32로 변경
 - iii) mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225]

Table of Contents

- Description
- Evaluation
- Random Seed 고정
- Model Summary
- 조교 계정

+ Add Section

<https://www.kaggle.com/t/49433a54b7864351a932c6ba1ee55aca>