8장 | 성능 최적화

WEEK	13주차

8.1 성능 최적화

8.1.1 데이터를 사용한 성능 최적화

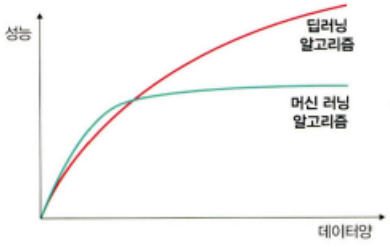
= 많은 데이터를 수집하는 것

△ 데이터 수집이 여의치 않은 상황 - 데이터 생성하는 방법도 고려

- **최대한 많은 데이터 수집하기**

  일반적으로 딥러닝이나 머신 러닝 알고리즘은 데이터양이 많을수록 성능이 좋음

그림 8-1 데이터와 딥러닝, 머신 러닝 알고리즘의 성능 비교



- **데이터 생성하기**

  많은 데이터를 수집할 수 없을 때 데이터를 만들어 사용

- **데이터 범위(scale) 조정하기**

  활성화 함수 - 시그모이드 → 데이터셋 범위 0~1

  활성화 함수 - 하이퍼볼릭 탄젠트 → 데이터셋 범위 -1~1

- **정규화, 규제화, 표준화**

8.1.2 알고리즘을 이용한 성능 최적화

유사한 용도의 알고리즘들을 선택하여 모델을 훈련시켜 보고 최적의 성능을 보이는 알고리즘을 선택

머신 러닝 - 데이터 분류를 위해 SVM, K-최근접 이웃 알고리즘들

시계열 데이터 - RNN, LSTM, GRU 등

8.1.3 알고리즘 튜닝을 위한 성능 최적화

성능 최적화 하는 데 가장 많은 시간이 소요되는 부분

모델을 하나 선택하여 훈련시키려면 다양한 하이퍼파라미터를 변경하면서 훈련시키고 최적의 성능을 도출

선택할 수 있는 하이퍼파라미터

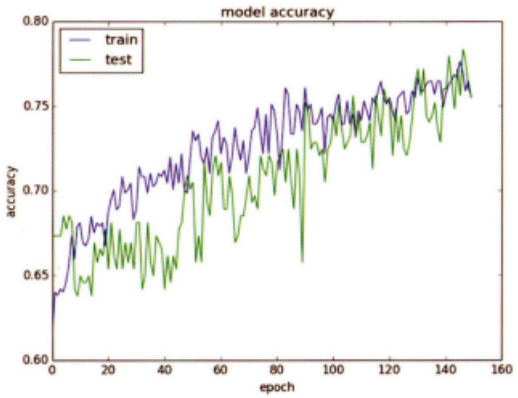
- **진단**

  성능 향상이 어느 순간 멈추었다면 원인을 분석할 필요함

문제를 진단하는 데 사용할 수 있는 것이 모델에 대한 평가

평가 결과를 바탕으로 모델이 과적합인지 혹은 다른 원인으로 성능 향상에 문제가 있는지에 대한 인사이트를 얻을 수 있음

❯ 그림 8-2 알고리즘 성능 진단



e.g.) 훈련 성능이 검증보다 눈에 띄게 좋을 경우 → 과적합 의심
⇒ 규제화

훈련과 검증 결과가 모두 성능이 좋지 않을 경우 → 과소적합 의심
⇒ 네트워크 구조를 변경 / 훈련 늘리기 위해 에포크 수 조정

훈련 성능이 검증을 넘어서는 변곡점
⇒ 조기 종료 고려

- **가중치**

  초깃값 - 작은 난수

  오토인코더 같은 비지도 학습을 이용하여 사전 훈련을 진행한 후 지도 학습을 진행

- **학습률**

  모델의 네트워크 구성에 따라 다름

  → 초기에 매우 크거나 작은 임의의 난수를 선택

  → 학습 결과를 보고 조금씩 변경
  네트워크 계층 多 ⇒ 학습률 ↑ / 네트워크 계층 少 ⇒ 학습률 ↓

- **활성화 함수**

  활성화 함수를 변경할 때 손실 함수도 함께 변경해야 하는 경우 多 → 신중해야 함

- **배치와 에포크**

  최근 딥러닝 트렌드 : 큰 에포크 작은 배치 사용

  △ 적절한 배치 크기를 위해 훈련 데이터셋 크기와 동일하게 하거나 하나의 배치로 훈련 시켜보는 등 다양한 테스트 진행하는 것이 좋음

- **옵티마이저 및 손실 함수**

  일반적으로 확률적 경사 하강법 사용 多

  네트워크 구성에 따라 차이 있지만 아담(Adam)이나 알엠에스프롭(RMSProp) 등도 좋은 성능을 보임

  다양한 옵티마이저와 손실 함수 적용해 보고 성능이 최고인 것을 선택

- **네트워크 구성**

  = 네트워크 토폴로지(topology)

  쉽게 알 수 있는 부분이 아니기 때문에 네트워크 구성을 변경해 가면서 성능 테스트해야 함

8.1.4 앙상블을 이용한 성능 최적화

앙상블 = 모델을 두 개 이상 섞어서 사용하는 것 → 성능 향상에 도움 됨

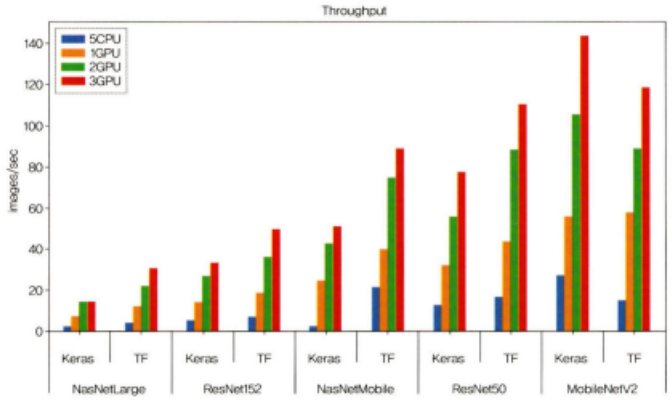
알고리즘 튜닝을 위한 성능 최적화 방법은 하이퍼파라미터에 대한 경우의 수를 모두 고려해야 함
→ 모델 훈련이 수십 ~ 수백 번 필요

8.2 하드웨어를 이용한 성능 최적화

기존 CPU가 아닌 GPU를 이용하여 딥러닝에서 성능 최적화

8.2.1 CPU와 GPU 사용의 차이

그림 8-3 CPU와 GPU 성능 비교[1]



CPU 다섯 개를 동시에 돌려도 GPU 한 개보다 성능이 좋지 못함
→ 왜 CPU와 GPU 성능 차이 발생할까?
⇒ CPU와 GPU는 개발된 목적이 다르고, 그에 따라 내부 구조도 다르기 때문

CPU 구성 : **ALU**, **Control**, **Cache**
→ CPU는 명령어가 입력되는 순서대로 데이터를 처리하는 직렬 처리 방식
= 한 번에 하나의 명령어만 처리하기 때문에 연산을 담당하는 ALU 개수가 많을 필요 X

GPU 구성 : 더 **낮은 비중**의 **캐시 메모리**, 더 **많은** 개수의 **ALU**

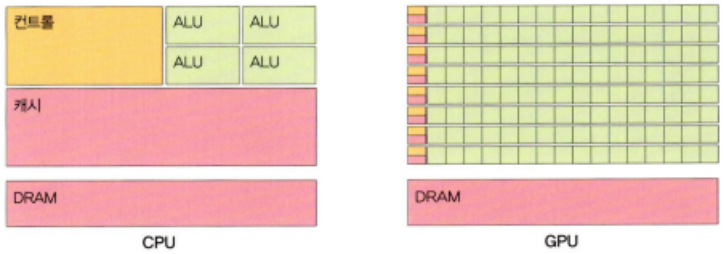
GPU(Graphics Processing Unit) 개발 목적 : 서로 다른 명령어를 동시에 병렬적으로 처리
→ 성능에 부담 X

= GPU는 많은 ALU로 구성되어 있기 때문에 여러 명령을 동시에 처리하는 병렬 처리 방식에 특화

하나의 코어에 ALU 수백 ~ 수천 개 장착
→ 3D 그래픽 작업 등을 빠르게 수행
→ △ 개별적 코어 속도는 CPU가 GPU보다 훨씬 빠름

= CPU가 적합한 분야가 따로 있고, GPU가 적합한 분야가 따로 있음

❯ 그림 8-4 CPU와 GPU의 구조 비교



순차적 연산일 때 CPU가 적합

△ 역전파처럼 복잡한 미적분은 병렬 연산을 해야 속도가 빨라짐

병렬 처리는 복잡한 연산이 수반되는 딥러닝에서 속도와 성능을 높여 주는 주요 요인이 될 수 있음

8.2.2 GPU를 이용한 성능 최적화

colab 이용하므로 건너뜀

8.3 하이퍼파라미터를 이용한 성능 최적화

하이퍼파라미터를 이용한 성능 최적화의 추가적인 방법 : 배치 정규화, 드롭아웃, 조기 종료

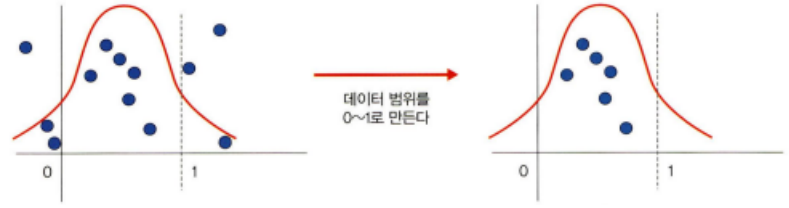
8.3.1 배치 정규화를 이용한 성능 최적화

유사한 의미로 사용되는 용어들

정규화(normalization)

데이터 범위를 사용자가 원하는 범위로 제한하는 것

❯ 그림 8-31 정규화



정규화는 각 특성 범위(스케일)을 조정한다는 의미로 특성 스케일링(feature scaling)이라고도 함

스케일 조정을 위해 MinMaxScaler() 기법을 사용하므로 수식은 다음과 같음

$$\frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

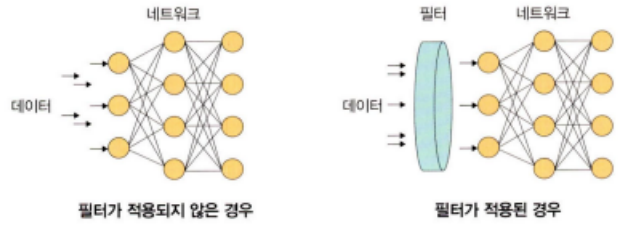
(x: 입력 데이터)

규제화(regularization)

모델 복잡도를 줄이기 위해 제약을 두는 방법

제약은 데이터가 네트워크에 들어가기 전에 필터를 적용한 것이라고 생각하면 됨

▼ 그림 8-32 규제화



왼쪽 그림은 필터가 적용되지 않을 경우 모든 데이터가 네트워크에 투입되지만
오른쪽 그림은 필터로 걸러진 데이터만 네트워크에 투입되어 빠르고 정확한 결과를 얻을 수 있음

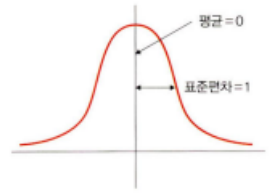
규제를 이용하여 모델 복잡도를 줄이는 방법 → 드롭아웃 / 조기 종료

표준화(standardization)

기존 데이터를 평균 0, 표준편차 1인 형태의 데이터로 만드는 방법

다른 표현 : 표준화 스칼라(standard scaler) 또는 z-스코어 정규화(z-score normalization)

▼ 그림 8-33 표준화



평균을 기준으로 얼마나 떨어져 있는지 살펴볼 때 사용함

보통 데이터 분포가 가우시안 분포를 따를 때 유용한 방법으로 다음 수식을 사용

$$\frac{x-m}{\sigma}$$

(σ: 표준편차, x: 관측 값(입력 값), m: 평균)

배치 정규화(batch normalization)

데이터 분포가 안정되어 학습 속도를 높일 수 있음

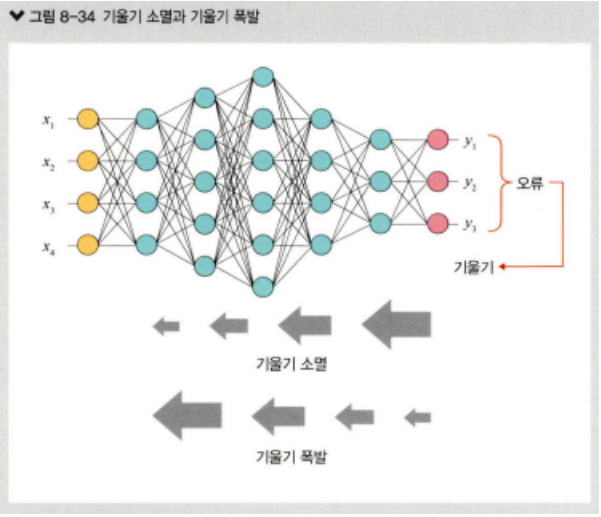
기울기 소멸(gradient vanishing)이나 기울기 폭발(gradient exploding) 같은 문제를 해결하기 위한 방법

일반적으로 기울기 소멸이나 폭발 문제를 해결하기 위해 손실 함수로 렐루를 사용하거나 초깃값 튜닝, 학습률 조정 등을 수행함

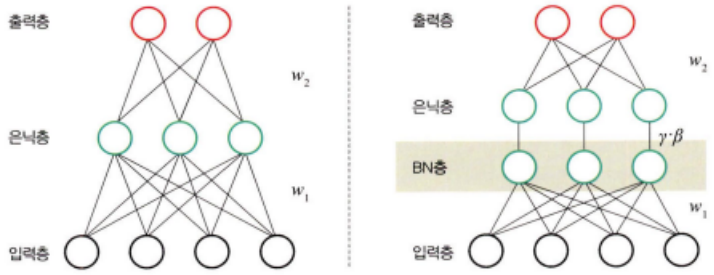
Note | 기울기 소멸과 기울기 폭발

- **기울기 소멸** : 오차 정보를 역전파시키는 과정에서 기울기가 급격히 0에 가까워져 학습이 되지 않는 현상
- **기울기 폭발** : 학습 과정에서 기울기가 급격히 커지는 현상

❯ 그림 8-34 기울기 소멸과 기울기 폭발



❯ 그림 8-35 배치 정규화



배치 정규화가 소개된 논문에 따르면 기울기 소멸과 폭발 원인은 내부 공변량 변화 때문

= 네트워크의 각 층마다 활성화 함수가 적용되면서 입력 값들의 분포가 계속 바뀌는 현상

⇒ 분산된 분포를 정규분포로 만들기 위해 표준화와 유사한 방식을 미니 배치에 적용하여 평균은 0으로, 표준편차는 1로 유지하도록 하며, 수식은 다음과 같음

$$\mu\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \quad \cdots\cdots ①$$

$$\sigma^2\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu\beta)^2 \quad \cdots\cdots ②$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu\beta}{\sqrt{\sigma^2\beta + \epsilon}} \quad \cdots\cdots ③$$

$$y_i \leftarrow \gamma\hat{x}_i + \beta \Leftrightarrow BN_{\gamma,\beta}(x_i) \quad \cdots\cdots ④$$

$$\left(\begin{array}{l} \text{입력: } \beta=\{x_1, x_2, \cdots, x_n\} \\ \text{학습해야 할 하이퍼파라미터: } \gamma, \beta \\ \text{출력: } y_i=BN_{\gamma,\beta}(x_i) \end{array}\right)$$

① 미니 배치 평균을 구함
② 미니 배치의 분산과 표준편차를 구함
③ 정규화를 수행함
④ 스케일을 조정(데이터 분포 조정)함

→ 매 단계마다 활성화 함수를 거치면서 데이터셋 분포가 일정해지기 때문에 속도를 향상시킬 수 있음

단점

1) 배치 크기가 작을 대는 정규화 값이 기존 값과 다른 방향으로 훈련될 수 있음
2) RNN은 네트워크 계층별로 미니 정규화를 적용해야 하기 때문에 모델이 더 복잡해지면서 비효율적일 수 있음

→ 이러한 문제들을 해결하기 위한 가중치 수정, 네트워크 구성 변경 등을 수행해야 함

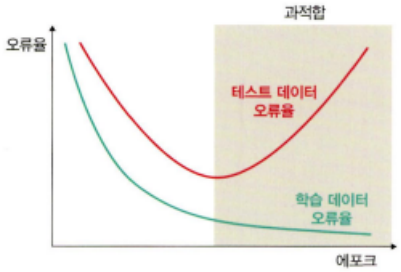
△ 배치 정규화를 적용하면 적용하지 않았을 때보다 성능이 좋아지기 때문에 많이 사용됨

8.3.2 드롭아웃을 이용한 성능 최적화

과적합 = 훈련 데이터셋을 과하게 학습하는 것

훈련 데이터셋은 실제 데이터셋의 부분집합이므로, 훈련 데이터셋에 대해 훈련을 계속한다면 오류는 줄어들지만 테스트 데이터셋에 대한 오류는 어느 순간부터 증가함 = 과적합된 모델

❤ 그림 8-36 훈련과 오류율

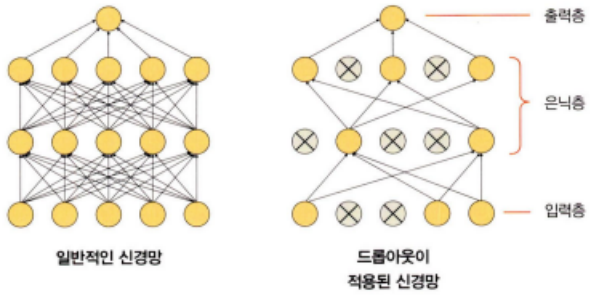


드롭아웃(dropout)이란?
훈련할 때 일정 비율의 뉴런만 사용하고, 나머지 뉴런에 해당하는 가중치는 업데이트하지 않는 방법
매 단계마다 사용하지 않는 뉴런을 바꾸어 가며 훈련시킴

= 노드를 임의로 끄면서 학습하는 방법으로, 은닉층에 배치된 노드 중 일부를 임의로 끄면서 학습함

꺼진 노드는 신호를 전달하지 않으므로 지나친 학습을 방지하는 효과

❤ 그림 8-37 드롭아웃



드롭아웃이 적용된 신겨앙은 일부 노드들은 비활성화되고 남은 노드들로 신호가 연결되는 신경망 형태를 띄고 있음

어떤 노드를 비활성화할지는 학습할 때마다 무작위로 선정되며, 테스트 데이터로 평가할 때는 노드들을 모두 사용하여 출력하되 노드 삭제 비율(드롭아웃 비율)을 곱해서 성능을 평가

훈련 시간이 길어지는 단점
△ 모델 성능 향상을 위해 상당히 자주 쓰는 방법

8.3.3 조기 종료를 이용한 성능 최적화

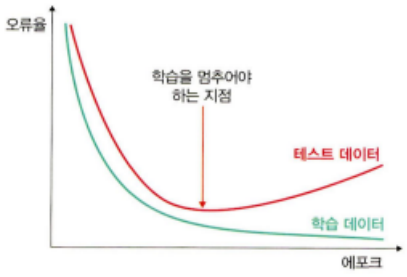
조기 종료(early stopping) = 뉴럴 네트워크가 과적합을 회피하는 규제 기법

훈련 데이터와 별도로 검증 데이터를 준비하고, 매 에포크마다 검증 데이터에 대한 오차(validation loss)를 측정하여 모델의 종료 시점을 제어

과적합이 발생하기 전까지 학습에 대한 오차(training loss)와 검증에 대한 오차 모두 감소하지만, 과적합이 발생하면 훈련 데이터셋에 대한 오차는 감소하는 반면 검증 데이터셋에 대한 오차는 증가

→ 조기 종료는 검증 데이터셋에 대한 오차가 증가하는 시점에서 학습을 멈추도록 조정함

그림 8-50 조기 종료



조기 종료는 학습을 언제 종료시킬지 결정할 분이지 최고의 성능을 갖는 모델을 보정하지는 않는다는 점에 주의

훈련 결과 비교

어떤 인수도 적용 X

에포크 100 지정했기 때문에 상당히 긴 시간의 학습 진행

그림 8-54 어떤 인수도 적용되지 않았을 때의 정확도

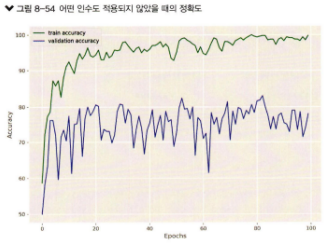
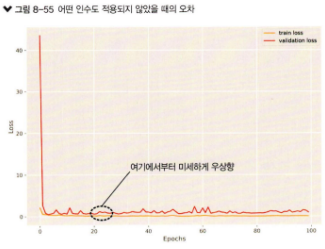


그림 8-55 어떤 인수도 적용되지 않았을 때의 오차



정확도 - 위아래로 많은 변동

오차 - 크게 다르지 않음. 특히 에포크 10 이후부터 검증 데이터셋에 대한 오차가 미세한 차이로 우상향

= 모델이 과적합되기 시작

→ 학습률 값 줄여야

학습률 감소 인수 적용

지정된 에포크 100 모두 수행되기 때문에 상당히 긴 시간 필요

그림 8-56 학습률 감소에 대한 인수가 적용되었을 때의 정확도

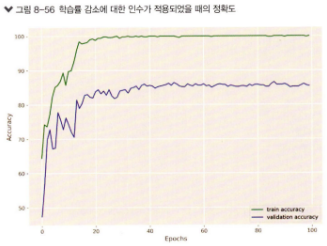
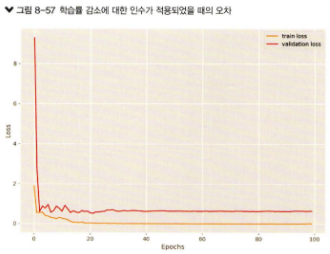


그림 8-57 학습률 감소에 대한 인수가 적용되었을 때의 오차



정확도 - 완만한 곡선 형태 + 훈련 종료 시점의 검증 데이터셋 정확도 높음

오차 - 그래프 우상향 현상 X

= 학습 스케줄러가 성능 향상에 어느 정도 기여함

조기 종료 인수 적용

5번째 에포크부터 조기 종료 수행

그림 8-58 조기 종료에 대한 인수가 적용되었을 때의 정확도

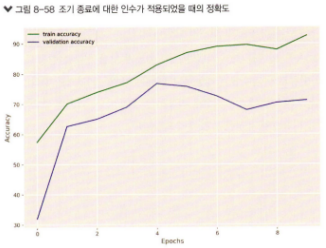
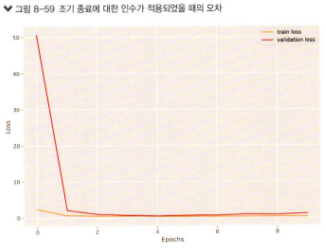


그림 8-59 조기 종료에 대한 인수가 적용되었을 때의 오차



5번째 에포크부터 조기 종료가 수행되고 있기 때문에 실제로 학습은 네 번째 에포크까지만 수행되었다고 할 수 있음

★ 조기 종료가 항상 성능에 좋은 영향을 미치는 것은 아님! 조기 종료로 인해 모델이 제대로 학습하지 못할 수 있음

△ 자세히 관찰하면 검증 데이터셋에 대한 오차가 에포크 20 정도에서도 정체되기 시작

= 에포크 30 정도만 수행해도 모델 훈련에 여전히 좋은 결과 얻을 수 있을 것

분명한 것은 **학습률 스케줄러를 이용한 학습률 조정 기법과 조기 종료가 모델 성능을 향상시키는 데는 도움이 된다**는 것!

단 **모든 모델에 일괄적으로 적용하는 것이 아닌, 기존의 출력된 그래프를 해석해서 어떤 성능 기법을 적용할지 결정하는 것이 중요**