8장 성능 최적화

생성일	@2024년 12월 16일 오전 11:33
주차	
완료여부	☐

8.1 성능 최적화

8.1.1 데이터를 사용한 성능 최적화

최대한 많은 데이터 수집하기: 데이터 양이 많을 수록 성능이 좋기 대무에 간으한 많은 데이터를 수집해야함

데이터 생성하기: 많은 데이터를 수집할 수 없다면 데이터를 만들어서 사용해야 함

데이터 범위 조정하기: 활성화함수로 sigmoid를 사용한다면 데이터셋 범위를 0-1 사이 값을 갖도록, hyper tan함수를 사용한다면 데이터셋 범위를 -1~1 사이 값을 갖도록 조정해야 함

8.1.2 알고리즘을 이용한 성능 최적화

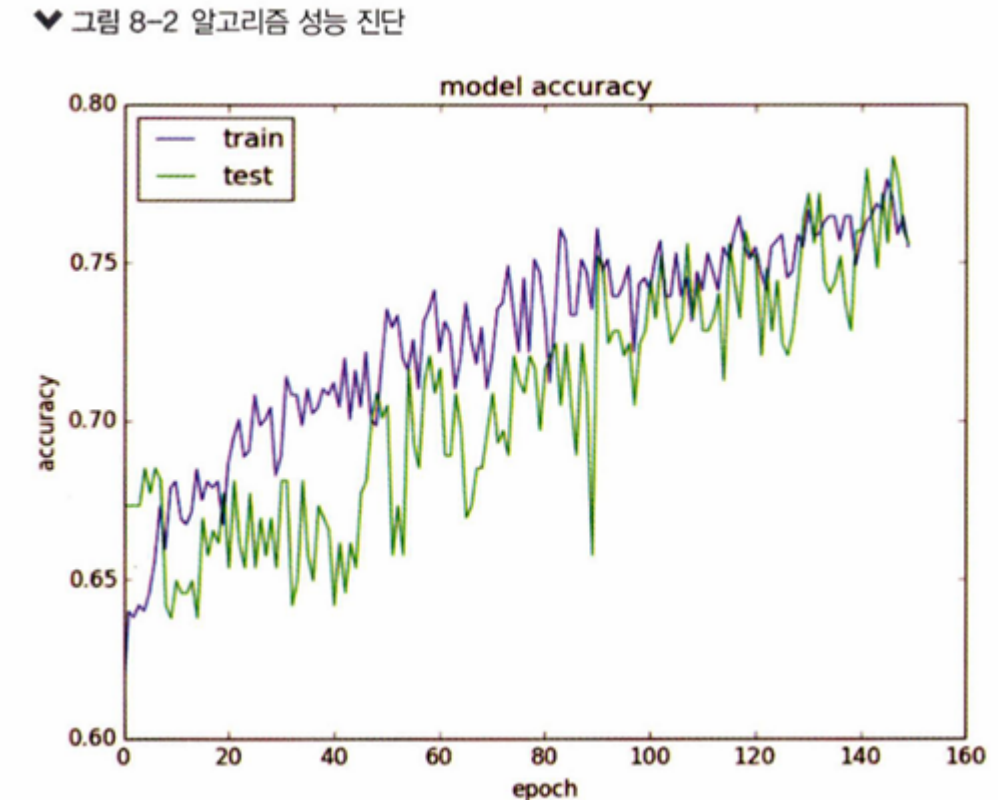
유사한 용도의 알고리즘들을 모두 훈련시켜보고 최적의 성능을 보이는 알고리즘을 선택해야 함

8.1.3 알고리즘 튜닝을 위한 성능 최적화

hyperparameter를 변경하면서 최적의 성능을 내야함

이때 성능 향상이 어느 순간 멈췄다면 원인을 분석해야 함. 모델 평가를 통해 overfitting이 문제인지 혹은 다른 원인이 있는지에 대한 insight를 얻어야 함

그림 8-2 알고리즘 성능 진단



train에서 성능이 test보다 눈에 띄게 좋다면 overfitting이 문제일 수도, 이것을 규제화를 통해 해결이 가능함

모두 성능이 좋지 않다면 underffiting 문제일 수도, epoch 수를 늘리거나 network 구조를 변경해 해결이 가능함

이외에 hyperparm : 가중치(초기 값은 작은 난수를 사용하지만 사전 훈련을 진행한 후 정보를 얻어 쓸 수도 있음), 학습률, 활성화함수(손실 함수도 함께 변경해야 함), 배치 와 에포크, 네트워크 구성 (topology 라고도 함, 은닉츠으이 뉴런의 갯수를 늘리거나 계층을 늘리되 뉴런의 갯수를 줄이는 방식으로)

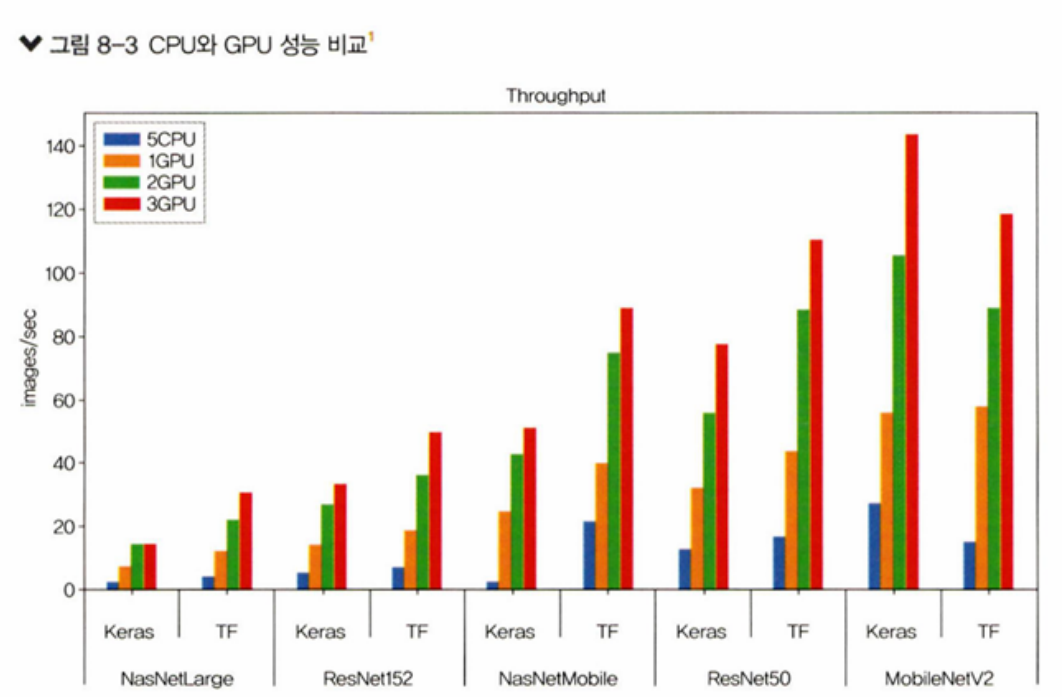
8.1.4 앙상블을 이용한 성능 최적화

두 개 이상의 모델을 섞어서 사용하여 하이퍼파라미터 튜닝에 대한 시간을 줄일 수 있음

8.2 하드웨어를 이용한 성능 최적화

기존 CPU가 아닌 GPU를 사용하여 성능을 높이는 방법

그림 8-3 CPU와 GPU 성능 비교[1]



CPU는 명령어가 입력되는 순서대로 데이터를 처리하는 직렬 처리 방식으로 연산을 담당하는 갯수가 많지 않음. 하지만 GPU는 병렬 처리가 가능하고 연산을 담당하는 ALU의 갯수가 많이 대문에 CPU 보다 빠르게 수행할 수 있음

개별적 코어속도는 CPU가 빠르지만 데이터를 벡터로 변환한 수 연산을 수행하는 딥러닝의 경우에는 GPU가 더 성능을 잘 냄

8.2.2 GPU를 이용한 성능 최적화

CUPA 란 NVIDIA에서 개발한 GPU 개발 툴로 CUDA를 사용해서 병렬처리 계산이 가능함.

표 8-1 GPU 모델 정보

구분	GEFORCE GTX 1080	GEFORCE GTX 1070	GEFORCE GTX 1060(3GB)	GEFORCE GTX 1050
Core	2560	1920	11512	640
Clock(MHz)	1607	1506	1506	1354
Memory	11GB	8GB	6GB	4GB
Memory Speed	10Gbps	8Gbps	8Gbps	7Gbps

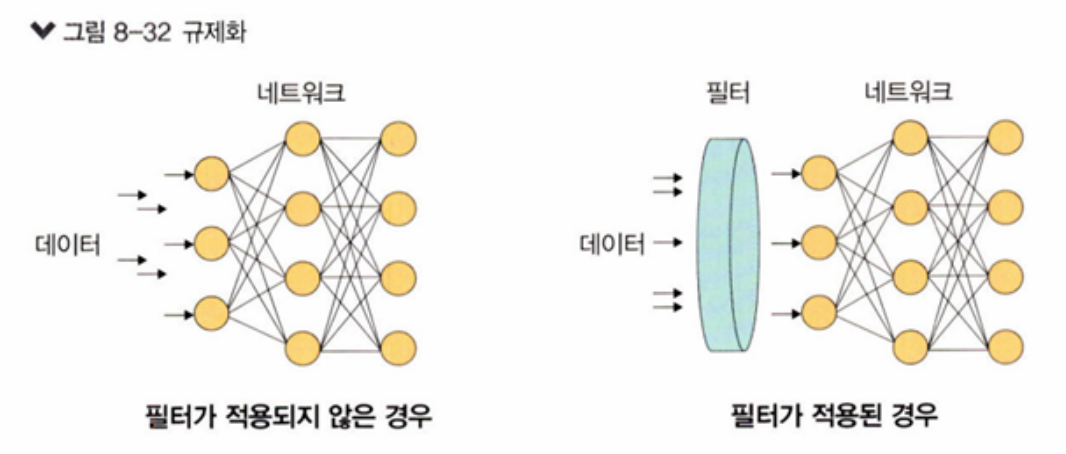
8.3 하이퍼파라미터를 이용한 성능 최적화

8.3.1 배치 정규화를 이용한 성능 최적화

정규화란 데이터의 범위를 사용자가 원하는 대로 제한하는 것을 의미함. feature를 scale 한다는 의미로 feature scaling이라고도 함

이 중 MinMaxScaler는 데이터의 min값과 max값을 이용하여 0-1 사이 값으로 조정하는 기법임

규제화란 모델의 복잡도를 줄이기 위해 제약을 두는 것인데, 이때 제약은 데이터가 네트워크에 들어가기 전에 필터를 적용한 것임

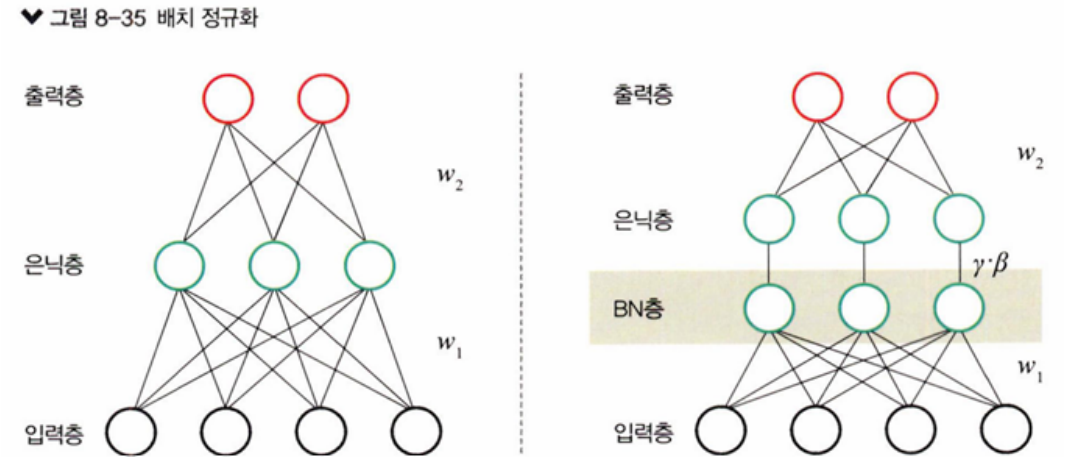


규제화의 종류로는 드롭아웃, 조기종료 등이 있음

표준화란 기존 데이터를 평균 0, 표준편차 1 인 표준정규분포를 따르는 데이터를 만드는 방법임

데이터의 분포가 가우시안 분포를 따를 때 유용한 방법임

배치정규화란 기울기 소멸, 폭발 의 문제를 해결하기 위한 방법으로 이 원인은 논문에 따르면 internal covariance shift 때문으로 네트워크의 각 층마다 활성화 함수를 적용하는데 입력값의 분포가 계속 바뀌기 때문임. 따라서 interal covariance를 정규분포로 만들기 위해 minibatch에 표준화와 유사한 방식을 상요해 분포의 range를 지정해줌

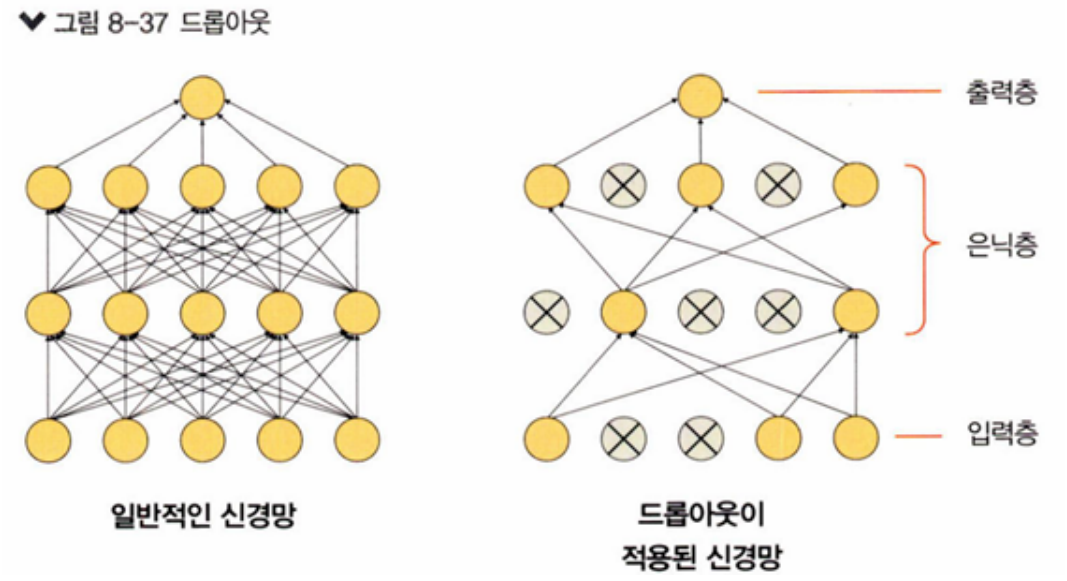


이를 통해 매 단계마다 활성화함수를 거치면서 데이터셋의 분포가 일정해져 속도를 향상시킬 수 있음

하지만 단점으로는, 배치 크기가 작을 때는 정규화 값이 기존 값과 많이 달라질 수 있음 예를 들어 분산이 0일 경우 정규화 자체가 안됨. 또한 RNN과 같이 복잡한 모델은 계층별로 미니 정규화를 적용해야하기 때문에 오히려 비효율적일 수 있음. 이 문제를 해결하기 위해 다른 파라미터를 조정해야 하지만, 기본적으로 배치 정규화를 적용하면 적용하기 전보다 성능이 좋아짐

8.3.2 드롭아웃을 이용한 성능 최적화

overfitting를 해결하기 위해 훈련할 때 일정 비율의 뉴런만 사용하고 나머지 뉴런은 가중치 업데이트를 하지 않는 방식이 dropout으로 은닉층의 노드를 일부를 임의로 끄면서 학습을 진행함. 이를 통해 지나친 학습을 방지할 수 있음. 훈련 시간이 증가한다는 단점이 있음



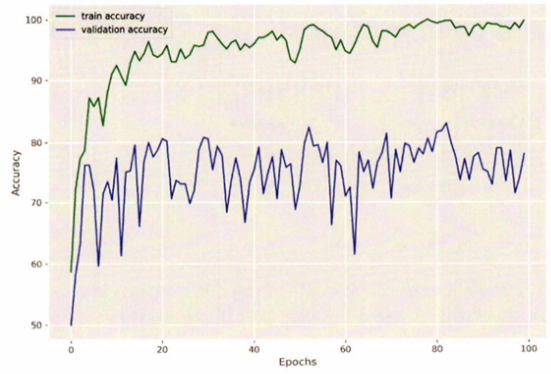
8.3.3 조기 종료를 이용한 성능 최적화

조기종료 early stopping은 과적합을 회피하는 규제 방법 중 하나로 훈련 데이터와 별도로 생성된 검증 데이터에서 매 애포크마다 validation loss를 측정하여 모델의 종료 시점을 제어함. overfitting 이후에는 train set에 대한 오차는 감소하지만 validation set에 대한 오차는 증가함. 조기종료는 validation set에 대한 오차가 증가하는 시점에서 학습을 멈추도록 조정함

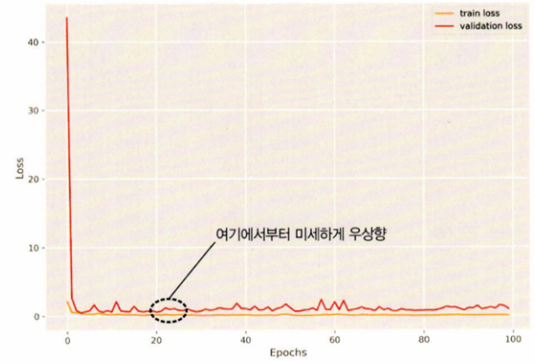
cpu를 사용하기 때문에 교재에 있는 epoch=100 이 아닌 epoch=5로 실행

따라서 교재에 있는 그래프를 통해 결과를 확인

❯ 그림 8-54 어떤 인수도 적용되지 않았을 때의 정확도

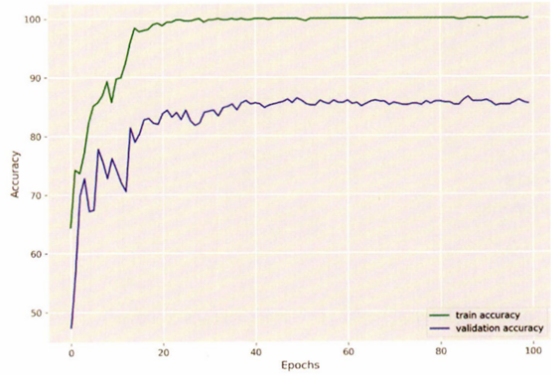


❯ 그림 8-55 어떤 인수도 적용되지 않았을 때의 오차

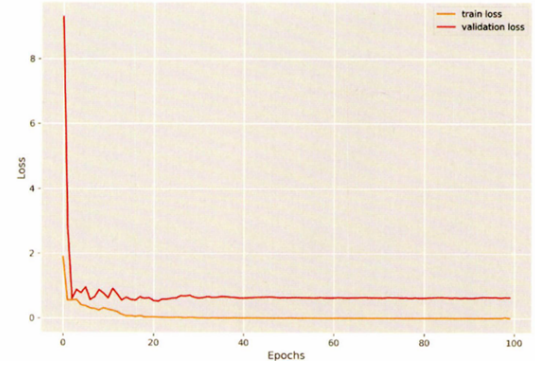


정확도 그래프의 많은 변동이 있는 것이 확인됨 + 오차에 대한 그래프로 epoch 20 이후부터 약간씩 증가함 → overfitting을 의미함

❯ 그림 8-56 학습률 감소에 대한 인수가 적용되었을 때의 정확도



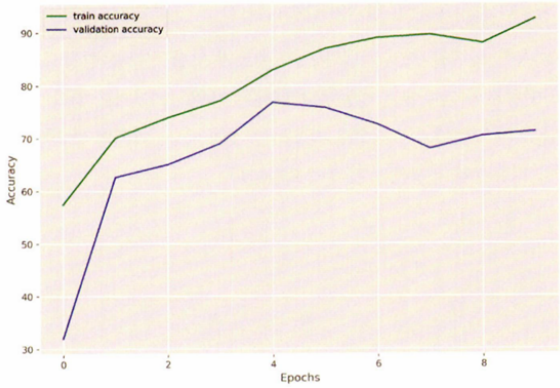
❯ 그림 8-57 학습률 감소에 대한 인수가 적용되었을 때의 오차



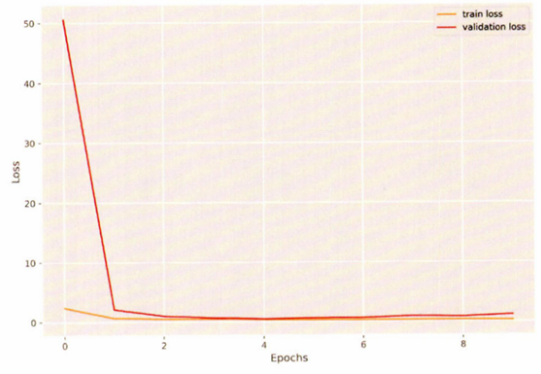
앞과 다르게 정확도 그래프가 보다 완만하고, 훈련이 종료된 시점에서 검증 데이터셋에 대한 정확도가 더 높은 것으로 확인이 됨

오차의 경우 epoch 40 정도에서 정체되기 시작한 걸 알 수 있음

❤ 그림 8-58 조기 종료에 대한 인수가 적용되었을 때의 정확도



❤ 그림 8-59 조기 종료에 대한 인수가 적용되었을 때의 오차



다섯번재 에포크부터 조기종료가 수행되고 있기 때무에 네번째 에포크까지만 수행되었다고 할 수 있음

조기종료로 모델이 제대로 학습하지 못할 가능성도 있음

따라서 그래프를 그려보면서 그래프에 대한 이해를 통해 적절한 성능 향상 방안을 적용하는 것이 중요함