개념정리 #13

딥러닝 파이토치 교과서 8장

8. 성능 최적화

8.1 성능 최적화

데이터를 사용한 성능 최적화

- **최대한 많은 데이터 수집하기** : 일반적으로 DL / ML 알고리즘은 데이터양이 많을수록 성능이 좋음
- **데이터 생성하기** : 데이터 증강 등
- **데이터 범위(scale) 조정하기**
    - 활성화 함수 - 시그모이드 (데이터셋 범위 0~1), 하이퍼볼릭 탄젠트 (데이터셋 범위 -1~1)
    - Normalization, Regularization, Standardization

알고리즘을 사용한 성능 최적화

수많은 알고리즘 중에 우리가 선택한 알고리즘이 최적의 알고리즘이 아닐 수도 있다. 따라서 유사한 용도의 알고리즘들을 선택해 모델을 훈련시켜 보고, 그 중 최적의 성능을 보이는 알고리즘을 선택할 필요가 있다.

알고리즘 튜닝을 위한 성능 최적화

모델을 하나 선택하여 훈련시키려면 다양한 하이퍼파라미터를 변경하면서 훈련시키고 최적의 성능을 도출해야 한다.

- **진단** : 모델에 대한 평가 — 성능 향상을 막는 overfitting 등의 원인에 대한 인사이트를 얻을 수 있음
    - Train 성능이 test보다 눈에 띄게 좋다면 : overfitting 의심

      → Regularization으로 해결

    - Train/test 성능이 모두 좋지 않다면 : underfitting 의심

      → 네트워크 구조 변경, training epoch 수 증가 등으로 해결

    - Train 성능이 test 성능을 넘어서는 변곡점이 있다면

      → early stopping 고려

- **가중치** : 초깃값 — 작은 난수 사용 or autoencoder 같은 비지도 학습을 이용해 사전 훈련을 진행한 후 지도 학습 진행
- **학습률** : 학습률은 모델의 네트워크 구성에 따라 다르기 때문에, 초기에 매우 크거나 작은 임의의 난수를 선택해 학습 결과를 보고 조금씩 변경해야함
    - 네트워크 계층이 많다면 : 학습률 높아야 함
    - 네트워크 계층이 적다면 : 학습률을 작게 설정해야 함
- **활성화 함수** : Activation function을 변경할 때 loss function도 함께 변경해야 하는 경우가 많음 → 다루고자 하는 데이터의 유형, 데이터로 어떤 결과를 얻고 싶은지에 대해 정확히 이해하지 못했다면 활성화 함수의 변경은 신중해야 함
    - 일반적으로 활성화 함수로 Sigmoid, Tanh 사용했다면 : 출력층에서는 Softmax, Sigmoid
- **배치와 에포크** : 일반적인 최근 딥러닝 트렌드 — 큰 에포크와 작은 배치 / 적절한 배치 크기를 위해 다양한 테스트를 진행하는 것이 좋음
- **옵티마이저 및 손실 함수**
    - 일반적으로 옵티마이저 : SGD 많이 사용함 + Adam, RMSProp
- **네트워크 구성** (= Network topology)
    - 넓은 네트워크 : 하나의 hidden layer에 뉴런을 여러 개
    - 깊은 네트워크 : layer 수를 늘리되 뉴런 개수는 줄임

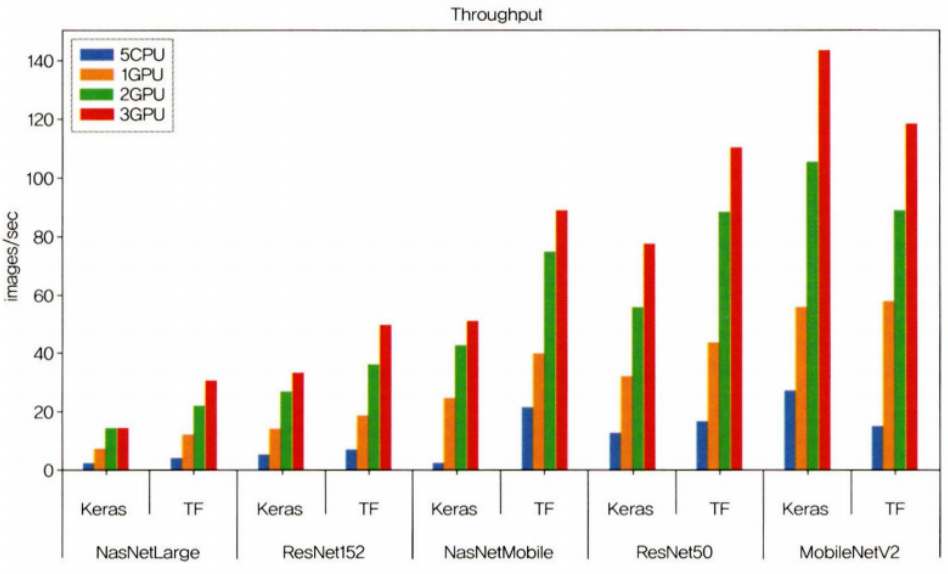
앙상블을 이용한 성능 최적화

앙상블 Ensemble : 모델을 두 개 이상 섞어서 사용하는 것

8.2. 하드웨어를 이용한 성능 최적화

CPU 대신 GPU를 이용해 성능 향상

CPU와 GPU 사용의 차이

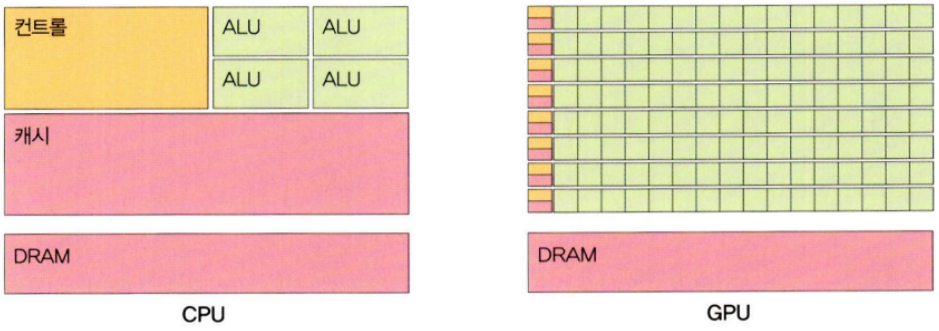


CPU

- ALU + control + 캐시
- 직렬 처리 방식 : 명령어가 입력되는 순서대로 데이터를 처리함

GPU

- 캐시 메모리 비중은 낮고, 연산을 수행하는 ALU 개수가 많아짐
- 병렬 처리 방식 : 서로 다른 명령어를 동시에 처리하도록 설계됨
- 하나의 코어에 ALU 수백~수천 개



개별적 코어 속도 : CPU >> GPU

행렬 연산 같은 순차적 재귀 연산 등의 직렬 연산에서는 CPU가 적합하다.

Backpropagation처럼 복잡한 미적분은 병렬 연산을 해야 속도가 빨라지므로 GPU가 적합하다.

⇒ 딥러닝은 수백~수천만 개의 데이터를 벡터로 변환 후 복잡한 연산을 수행하므로 GPU를 사용해 모델 트레이닝 시간을 효율적으로 단축할 수 있다.

GPU를 이용한 성능 최적화

CUDA : Computed Unified Device Architecture. NVIDIA에서 개발한 GPU 개발 툴

Google Colab에서는 GPU 환경을 클라우드로 제공해준다.

8.3. 하이퍼파라미터를 이용한 성능 최적화

배치 정규화를 이용한 성능 최적화

- **정규화 Normalization** : 데이터 범위를 사용자가 원하는 범위로 제한. Feature scaling이라고도 함.
  - `MinMaxScaler()` : $\frac{x-x_{\min}}{x_{\max}-x_{\min}}$ (x : input data)
- **규제화 Regularization** : 모델 복잡도를 줄이기 위해 제약을 두는 방법.
  - Dropout
  - Early stopping
- **표준화 Standardization** : 데이터를 평균 0, 표준편차 1인 형태로 변환하는 방법. Standard scaler 혹은 z-score normalization이라고도 함.
  - 평균을 기준으로 얼마나 떨어져 있는지 살펴볼 때 유용함
  - 보통 데이터 분포가 가우시안 분포를 따를 때 유용함
  - $\frac{x-m}{\sigma}$ (x : input data, m : mean, σ : std)

- **배치 정규화 Batch Normalization**

  : 데이터 분포가 안정되어 학습 속도를 높일 수 있음. Gradient vanishing/exploding 같은 문제를 해결하기 위한 방법.

  - Gradient vanishing/exploding problem의 원인 : 내부 공변량 변화(internal covariance shift, 네트워크의 각 층마다 활성화 함수가 적용되면서 입력 값들의 분포가 계속 바뀌는 현상)

    ⇒ 분산된 분포를 정규분포로 만들기 위해 standardization과 유사한 방식을 mini-batch에 적용해 평균은 0으로, 표준편차는 1로 유지하도록 함

1. $\mu\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$: 미니 배치 평균을 구한다
2. $\sigma^2\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu\beta)^2$: 미니 배치의 분산과 표준편차를 구한다
3. $\hat{x_i} \leftarrow \frac{x_i-\mu\beta}{\sqrt{\sigma^2\beta+\epsilon}}$: 정규화를 수행한다
4. $y_i \leftarrow \gamma\hat{x_i}+\beta \Leftrightarrow \mathrm{BN}_{\gamma,\beta}(x_i)$: 스케일을 조정한다

매 단계마다 활성화 함수를 거치면서 데이터셋 분포가 일정해짐

- 장점 : 속도 향상 가능
- 단점 :
  - 배치 크기가 작을 때는 정규화 값이 기존 값과 다른 방향으로 트레이닝될 수 있음 (e.g. 분산이 0이면 normalization 자체가 안 됨)
  - RNN는 네트워크 계층 별로 mini normalization을 적용해야 함 → 모델이 더 복잡해지고 비효율적

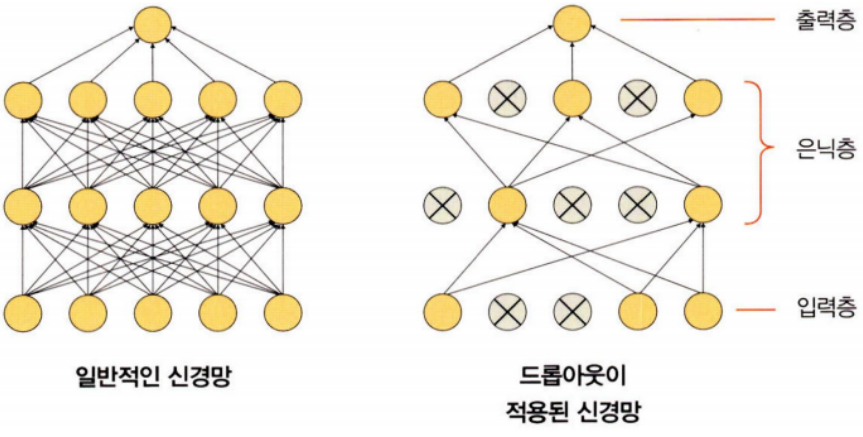
> 💡 **BN 사용하는 이유**
>
> Hidden layer에서 학습이 진행될 때마다 입력 분포가 변하면서 weight이 엉뚱한 방향으로 갱신되는 문제가 종종 발생하기 때문이다. 즉, 신경망의 층이 깊어질수록 학습할 때 가정했던 입력 분포가 변화해 엉뚱한 학습이 진행될 수 있는데, BN을 적용해서 입력 분포를 고르게 맞추어 줄 수 있다.

※ 배치 정규화 위치

FC과 Conv 뒤 / Activation funct. 앞

드롭아웃을 이용한 성능 최적화

드롭아웃 Dropout : 트레이닝 시 일정 비율의 뉴런만 사용하고, 나머지 뉴런에 해당하는 weight은 업데이트하지 않는 방법. Hidden layer에 배치된 노드 중 일부를 임의로 끄면서 학습함.

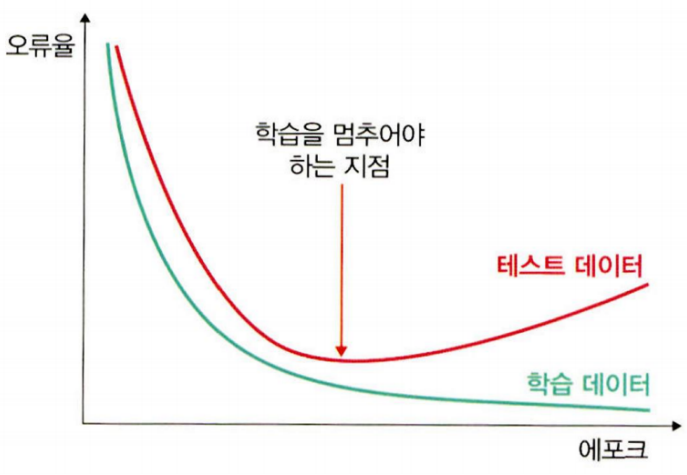


- 어떤 노드를 비활성화할지는 학습 때마다 무작위로 선정됨
- 테스트 데이터로 평가할 때는 모든 노드를 사용해 출력하되 노드 삭제 비율(드롭아웃 비율)을 곱해서 성능을 평가함
- 트레이닝 시간이 길어지는 단점이 있지만, 모델 성능을 향상하기 위해 —overfitting을 피하기 위해— 상당히 자주 쓰는 방법임

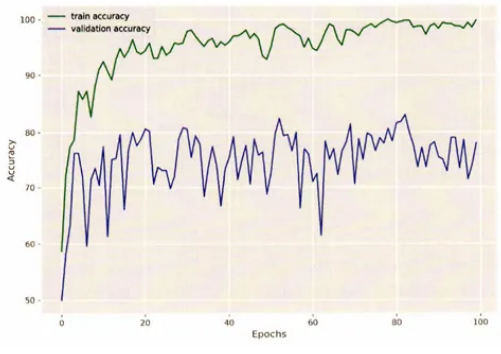
조기 종료를 이용한 성능 최적화

조기 종료 Early stopping : 뉴럴 네트워크가 overfitting을 회피하는 regularization 기법

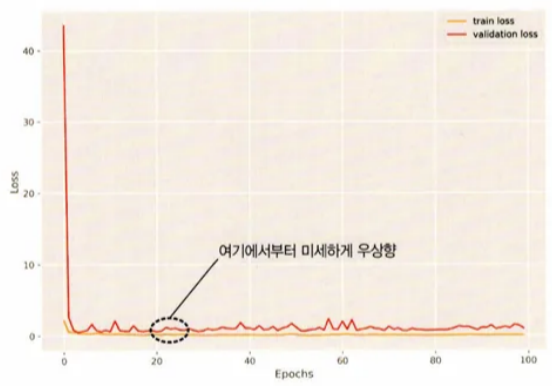
- 매 에포크마다 validation data에 대한 loss를 측정해 모델의 학습 종료 시점을 제어함
- Validation loss가 증가하는 시점에서 학습을 멈추도록 조정함
- 학습을 언제 종료시킬지 결정할 뿐 최고의 성능을 갖는 모델을 보장하지는 않음!



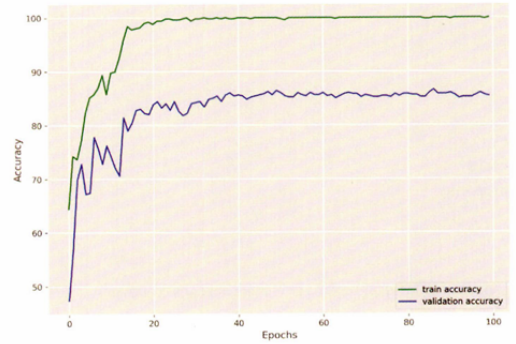
❣ 그림 8-54 어떤 인수도 적용되지 않았을 때의 정확도



❣ 그림 8-55 어떤 인수도 적용되지 않았을 때의 오차



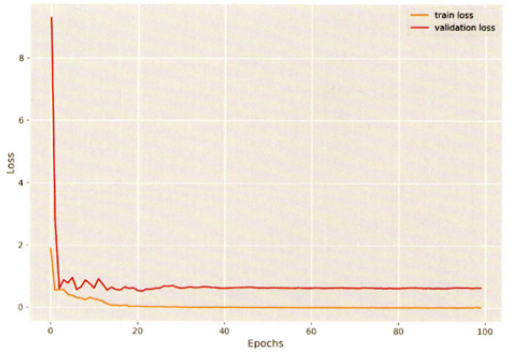
❯ 그림 8-56 학습률 감소에 대한 인수가 적용되었을 때의 정확도



⬆ Learning rate decay를 사용하면 accuracy 그래프가 완만한 곡선 형태를 보이며, 트레이닝이 종료된 시점의 valid set에 대한 accuracy도 높게 나타나고 있음

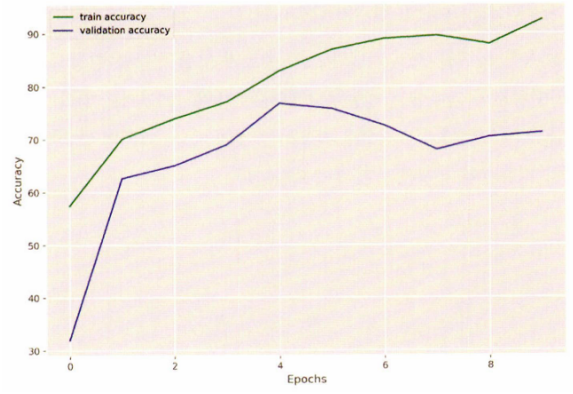
⇒ Learning rate scheduler가 성능 향상에 어느 정도 기여함!

❯ 그림 8-57 학습률 감소에 대한 인수가 적용되었을 때의 오차

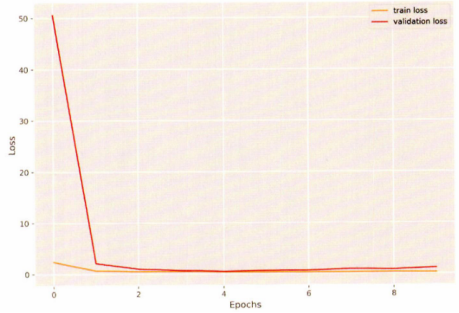


⬆ Valid set에 대한 loss가 에포크 20 정도에서 정체하고 더 이상 감소하지 않음

❯ 그림 8-58 조기 종료에 대한 인수가 적용되었을 때의 정확도



❯ 그림 8-59 조기 종료에 대한 인수가 적용되었을 때의 오차



Early stopping이 항상 성능에 좋은 영향을 미치는 것은 아님 — 모델이 제대로 학습하지 못할 수 있음

성능 향상보다는 cost 효율화