💛

8장

성능 최적화

8.1 성능 최적화

→ 다양한 방법이 존재하는데

많은 데이터 수집하기, 비슷한 알고리즘 중 적절한 것을 선택, 하이퍼 파라미터 튜닝, 앙상블 등이 존재.

8.2 하드웨어를 이용한 성능 최적화

딥러닝 이용시에 GPU를 사용하는 것. (cpu5개를 사용하는 것보다 gpu한개의 성능이 좋음…)

→ 개발된 목적이 다르기 때문이다.

CPU	GPU
입력 순으로 데이터 처리(직렬) → 연산담당 ALU 개수가 많다.	서로 다른 명령어를 동시에 처리(병렬) → 연산담당 ALU가 많고 3D 그래픽 작업 빨리 수행
개별 코어 속도 빠르다	CPU에 비해 개별 코어 속도 드리다
	역전파, 복잡한 딥러닝에서 필수

8.3 하이퍼파라미터를 이용한 성능 최적화

- 배치 정규화

기울기 소멸과 폭발 문제가 발생한다.

분산된 분포를 정규분포로 만들기 위해 표준화와 유사한 방식을 미니-배치에 적용하여 평균=0, 표준편차=1로 유지하도록 한다.

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

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

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

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

① 미니 배치 평균을 구합니다.

② 미니 배치의 분산과 표준편차를 구합니다.

③ 정규화를 수행합니다.

④ 스케일(scale)을 조정(데이터 분포 조정)합니다.

→ 위의 식을 이용하면 매 단계마다 활성화 함수를 거치면서 데이터 분포가 일정해져 속도 향상이 있지만 1) 배치 크기가 작을 경우, 다른 방향으로 훈련이 될 수 있다. 2) RNN의 경우 모델이 복잡해질 수 있다.

`nn.BatchNorm1d(48)`

→ 배치 정규화를 사용하여 입력 분포를 고르게 맞춰준다. 위치는 FC와 conv layer 뒤, 활성화 함수 앞에 위치한다.

- 드롭아웃

과적합: 훈련 데이터를 과하게 훈련, 테스트 데이터에서 오류 증가

→ 드롭아웃을 이용하여 일정 비율의 뉴런만을 바꾸어 가며 훈련시키고 가중치를 업데이트하여 임의로 노드를 끄면서 학습하는 것. (비활성할 노드는 무작위로 선정괴며 테스트 데이터에서는 노드를 모두 사용한다.)

`torch.nn.Dropout(0.2)`

- 조기 종료

과적합을 회피하는 규제 기법이다. + 학습률 조정

콜백함수(https://velog.io/@ko1586/Callback함수란-뭔데) 참고

- 1. 동기 : 하나의 요청이 오면 완료가 된 후 다음 요청을 실행하는 방식 - 순차적 로직흐름2. 비동기 : 어떤 요청이 오면 완료가 되기 전에 다음 요청을 실행하는 방식동시 효율적 처리 가능, 즉시 응답X 때문에 예상 밖 결과 나올수도 있음.,
- 콜백함수는 때로는 가독성이나 코드 재사용 면에서도 사용 된다.
- 비동기 방식으로 작성된 함수를 동기 처리하기 위해 필요 하다.