8강: 신경망 파트3 - 학습 및 평가

그래디언트 확인(gradient check)

- 분석적 그래디언트는 미적분학 지식을 통해 그래디언트 계산
- 수치적 그래디언트를 아래 식을 활용하여 근사적으로 그래디언트를 계산하는 방식

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x)}{h} \quad \text{(bad, do not use)}$$

- 그래디언트 확인이란 이 두 그래디언트를 비교하는 과정임
- 그래디언트 확인 과정은 생각보다 복잡하고 오류가 발생하기 쉬움

팁1: 중심 차분 공식(centered difference formula) 사용

- 수치적 그래디언트 계산 시 중심 차분 공식을 사용하는 것이 훨씬 더 잘 계산됨

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2h} \quad \text{(use instead)}$$

- 중심차분 공식은 그래디언트의 각 차원에 적용을 해야함
- 중심차분 형태는 각 차원 당 손실 함수를 2번 계산하게 함
- 계산비용은 2배인 대신 그래디언트 근사값이 훨씬 더 정확히 계산

팁2: 비교를 위해 상대 오차(relative error) 사용

- 상대 오차를 사용하는 것이 좋음
- 만약 $|f'_a - f'_n|$ 혹은 $|f'_a - f'_n|^2$ 이 크면 그래디언트 확인이 실패라고 판단하는 경우에 문제가 있음
- 둘의 차이가 10^{-4}인 경우 두 그래디언트가 1.0 근처이면 이 차이는 매우 적절한 차이이므로 두 그래디언트가 일치한다 생각
- 두 그래디언트가 모두 10^{-5} 이하라면 10^{-4}는 매우 큰 차이이므로 그래디언트 확인은 실패라고 판단될 것임
- 따라서 항상 상대 오차를 고려하는 것이 더 적절함

팁2: 비교를 위해 상대 오차(relative error) 사용

$$\frac{|f'_a - f'_n|}{\max(|f'_a|, |f'_n|)}$$

- 대칭성을 위해 분모에서는 f'_a, f'_n 값 두 개를 모두 활용함
- 두 값중 0인 것이 있는 경우 0으로 나눠지지 않아야하므로 두 값의 최대값으로 나누는 것이 선호됨
- 두 값 모두 0인 경우도 잘 체크해야함

팁2: 비교를 위해 상대 오차(relative error) 사용

- 상대오차 $> 10^{-2}$
  - 그래디언트가 잘못됐다는 것을 의미
- $10^{-2} >$ 상대오차$>10^{-4}$
  - 불편함을 느낌
- $10^{-4} >$ 상대오차
  - 일반적으로 꺾임점이 있는 목표에 대해서는 괜찮음
  - 꺾임점이 없는 경우(ex. tanh, softmax 사용) 10^{-4}는 너무 높음
- $10^{-7} >$ 상대오차
  - 만족

팁2: 비교를 위해 상대 오차(relative error) 사용

- 신경망이 깊을 수록 상대 오차는 더 커짐
- 10층 신경망 입력 데이터에 대해 그래디언트를 확인하는 경우 오류가 누적되어 10^{-2}의 상대 오차가 괜찮을 수도 있음
- 단일 미분 가능함수에 대한 10^{-2} 오차는 잘못된 그래디언트를 나타낼 가능성이 높음

팁3: 배정밀도(double precision) 사용

- 일반적인 실수는 그래디언트 확인을 계산할 때 단정밀도(single precision) 부동소수점을 사용하는 것
- 그래디언트 구현이 올바르더라도 상대오차가 높게 나타나는 경우 있음 (ex. 0.01)
- 배정밀도로 전환함으로써 상대오차가 급격이 떨어지는 경우가 종종 있음

팁4: 부동 소수점의 활성범위 주변에 머무르기

- 손실 함수를 배치 수로 나눌 때 매우 작은 숫자를 생성하는 경우 많은 수치적인 문제를 초래할 수 있음
- 수치적/분석적 그래디언트 비교 시 두 값이 극도로 작지는 않은지 확인하는 것이 좋음
- 매우 작아진다면, 일시적으로 손실함수를 스케일링하여 더 좋은 부동소수점 범위로 가져오는 것이 좋음
  - ex) 1.0 주변

팁5: 목적 함수의 꺾임 주의

- ReLU 함수, SVM 손실, maxout 뉴런 등의 함수는 꺾인 지점(즉, 미분 불가능한 지점)이 있음
  - ReLU = max(0,x)
  - maxout: $\max(w_1^T x + b_1, w_2^T x + b_2)$

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

- 이런 부분에서의 그래디언트 확인은 에러가 생길 수 있음
- ex) ReLU 함수의 경우 $x = 10^{-6}$에서의 그래디언트 계산 시 해석적 그래디언트는 0이지만 수치적 그래디언트는 0이 아닐 수 있음
- 많은 꺾임 함수들이 있기 때문에 이런 에러가 꽤 흔할 수 있음

팁5: 목적 함수의 꺾임 주의

- 손실을 계산할 때 꺾임이 교차되었는지 여부를 체크할 수 있음
- 손실 계산 시 max(x,y) 형태의 함수에서 어떤 것이 큰지도 추적함
- 즉, 순방향 패스 동안 x,y 중 어떤 것이 큰지에 대한 정보도 저장
- 그러면 수치적 그래디언트 계산 시 f(x+h), f(x-h) 계산에서 꺾임이 교차되었는지 체크할 수 있고, 꺾임이 교차된 경우라면 수치적 그래디언트가 정확하지 않을 것임

팁6: 몇 가지의 데이터 포인트만 사용

- 꺾임 문제를 해결하기 위한 하나의 방법은 이미지 수를 줄이는 것
- 꺾임을 포함하는 손실 함수는 이미지가 많을 수록 꺾임이 적게 발생
- 가령 2-3개의 이미지에 대해서만 그래디언트 확인을 수행하여도 그래디언트 계산 식에 이상 여부를 확인하는데 충분할 수 있음
- 아주 적은 수의 이미지를 사용하면 그래디언트 확인이 더 빠르고 효율적임

팁7: 스텝 크기 h에 주의

- 일반적으로 수치적 그래디언트 계산 시 h가 작을 수록 더 좋은 근사값을 얻음

$$[f(x+h) - f(x-h)]/2h$$

- 그렇지만 h가 매우 작아질 경우 수치 정밀도 문제로 더 좋은 근사값을 얻지 못할 수도 있음
- 때때로, 그래디언트 확인에 실패했을 때, h를 10^{-4} 혹은 10^{-6}으로 키우면 그래디언트 확인에 성공할 수 있음

팁8: "특성화"된 동작 모드에서 기울기 확인

- 단일 지점에서 그래디언트 확인이 성공하더라도 그래디언트가 전체적으로 올바로 구현된 건지는 확실치는 않음
- 또한, 랜덤 초기화는 파라미터 공간에서 "특성화"된 지점이 아닐 수 있고, 실제로 그래디언트가 올바로 구현되지 않았음에도 된 것처럼 보일 수 있음
- 예를 들어, 가중치 초기화가 매우 작은 SVM은 모든 이미지에 대해 거의 0의 점수를 할당하면, 일부 점수가 다른 점수보다 큰 특성화된 모드로 일반화되지 못할 수 있음
- 안전한 그래디언트 확인을 위해서는, 신경망이 먼저 잠시 학습하면서 손실이 감소하기 시작한 후에 그래디언트 확인을 해야함

팁9: 정규화가 데이터를 압도하지 않도록 함

- 데이터 손실이 정규화 손실을 압도하지 않도록 해야함
- 그렇지 않으면 데이터 손실의 잘못된 구현을 파악하지 못하게 할 수 있음
- 먼저, 정규화를 끄고 데이터 손실만 확인한 후, 독립적으로 정규화 항을 확인하는 것이 권장됨
- 정규화 항만 독립적으로 확인하는 방법
  - 데이터 손실을 포함 안 하도록 코드 수정
  - 정규화 강도를 늘려 그래디언트 확인에서의 큰 비중을 차지하도록 함

이성 검사 (sanity check)

- 시간이 많이 걸리는 훈련 과정 전에 초기화나 구현에 문제가 없는지 확인하는 것은 이성 검사(sanity check)라고 불림
- 방법1: 작은 파라미터로 초기화할 때 예상하는 손실을 얻는지 확인

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \qquad L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

  - CIFAR-10에서 소프트맥스 분류기의 경우 초기손실이 2.302, SVM ($\Delta = 1$ 인 경우) 의 경우 9일 것으로 예상됨

이성 검사 (sanity check)

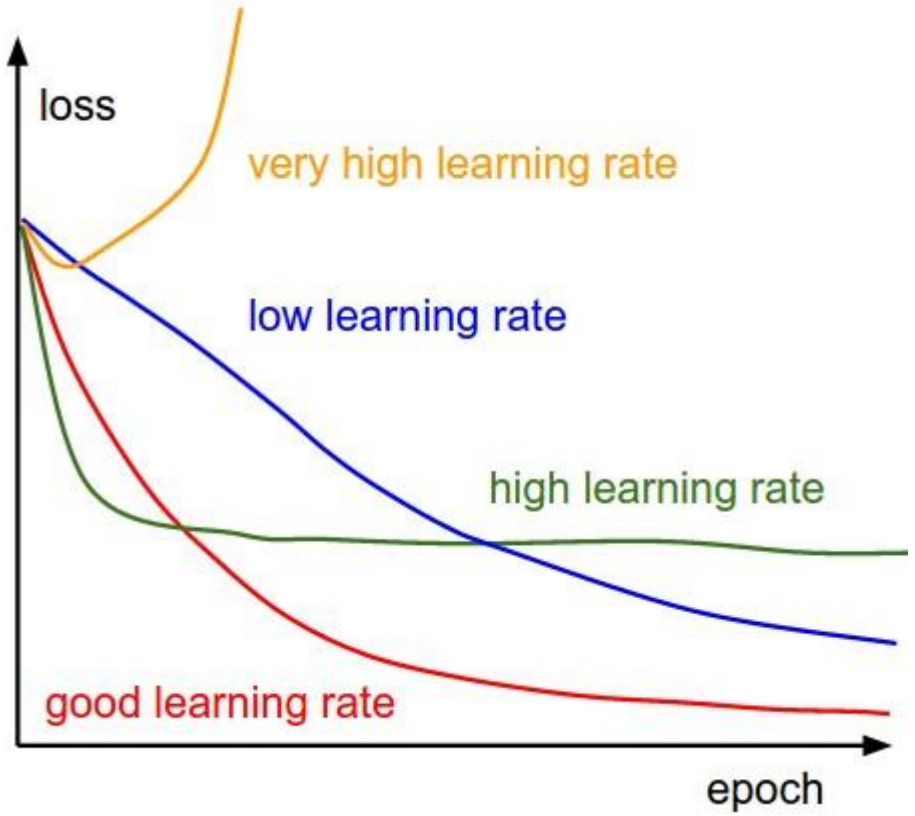
- 방법2: 정규화 강도를 늘릴 때 손실이 증가하는지 확인
- 방법3: 매우 작은 데이터셋에 과적합 시켜봄
  - 데이터의 아주 작은 부분(ex, 20개 이미지)에 대해 학습시키고 손실이 0에 도달하는지 확인해봄
  - 손실이 0이 되는 것을 방해할 수 있기 때문에 정규화는 0으로 설정
  - 작은 데이터셋에 대해 이 이성검사를 통과하지 못하면 전체 데이터셋으로 넘어가는 것은 가치가 없음

훈련 과정 중 모니터링

- 신경망 훈련 중 중요한 수치들을 모니터링 하여 그래프로 그리면 , 다양한 하이퍼파라미터 설정에 대한 직관을 얻고, 어떻게 하이퍼파라미터를 변경해야할지 파악하는데 사용할 수 있음
- 다음에 나올 그래프들은 x축이 에폭(epoch) 단위
- 에폭이란 학습 중 각 이미지를 대략 몇 번 볼 수 있는지를 측정
- 배치 크기의 설정에 따라 반복횟수가 달라지므로, 반복 횟수보다는 에폭을 모니터링하는 것이 더 좋음

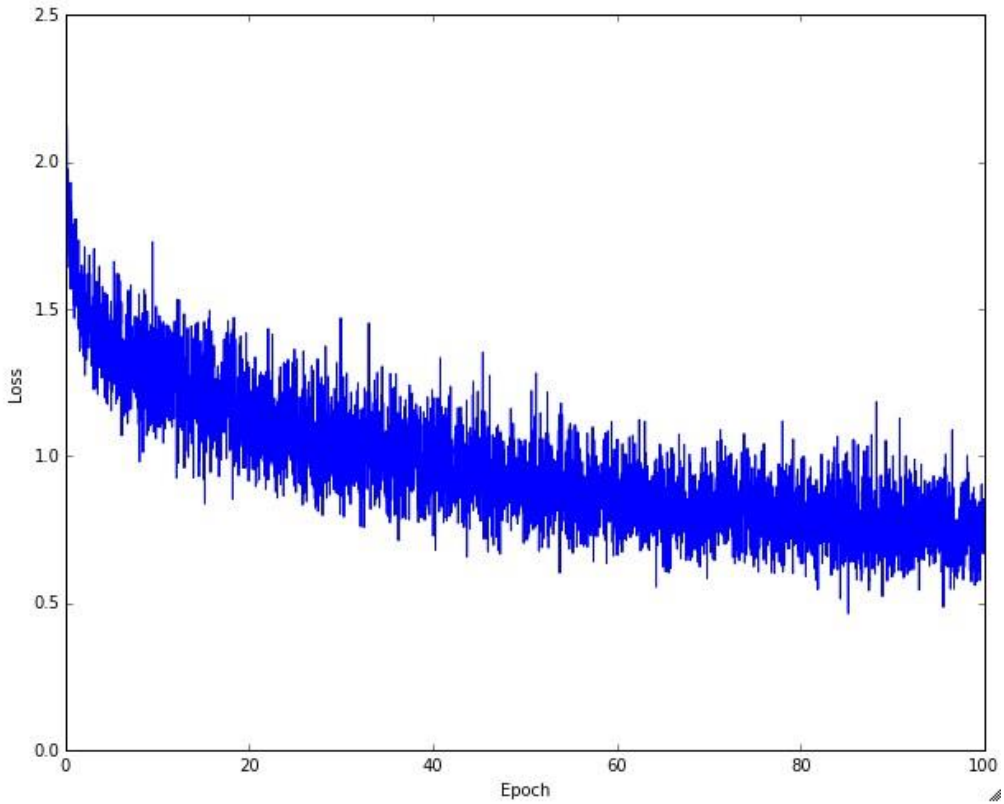
모니터링1: 손실

- 손실은 학습 중 추적하는 것이 중요한 수치 중 하나
- 낮은 학습률 사용 시 손실 그래프는 선형적임
- 높은 학습률 사용 시 더 지수적이지만 안 좋은 손실 값에 수렴할 수 있음
- 좋은 학습률은 어느 정도 빨리 손실이 감소하면서도 낮은 손실 값에 수렴함



모니터링1: 손실

- CIFAR-10 데이터셋에서 작은 신경망을 학습하는 동안의 일반적인 손실 그래프
- 각 배치의 손실 값들을 그래프로 그림
- 손실의 진동 정도는 배치 크기와 관련됨
- 배치 크기가 1이면 진동은 상대적으로 높음
- 배치 크기가 전체 데이터셋이면 진동은 최소화됨

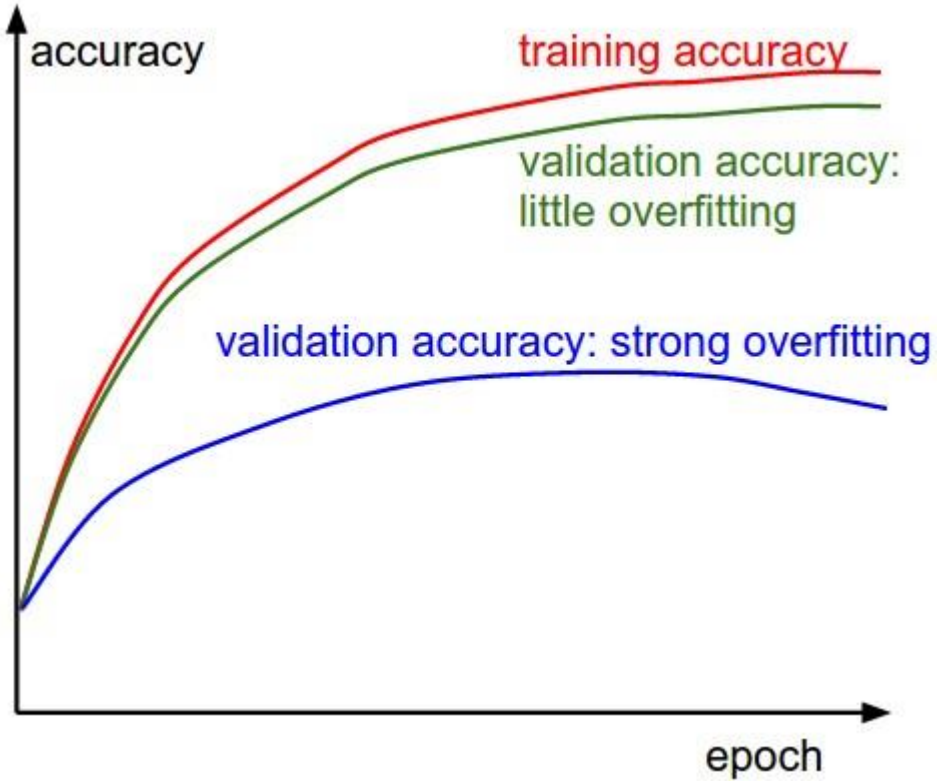


모니터링1: 손실

- 일부 사람들은 손실 함수를 로그 도메인에서 그래프를 그리는 것을 선호함
- 그래프가 약간 더 해석하기 쉬운 직선으로 나타나게 됨

모니터링2: 학습/검증 정확도

- 학습 정확도와 검증 정확도의 차이는 과적합의 정도를 나타냄
- 파란색 검증 정확도 곡선은 강한 과적합을 나타냄
- 과적합 해결을 위해서는 정규화 강도를 늘리거나 데이터셋을 더 수집할 수 있음
- 초록색 검증 정확도 곡선은 검증 정확도가 훈련 정확도를 꽤 잘 따라감



- 이 경우는 신경망 크기가 충분히 크지 않은 것이므로, 파라미터의 수를 늘려 신경망을 크게 만드는 것 고려

모니터링3: 업데이트 / 가중치값 비율

- 업데이트 크기와 값 크기의 비율을 모니터링할 수 있음
- 업데이트란 그래디언트에 학습률이 곱해진 것
- 이 비율은 모든 파라미터 세트에 대해 독립적으로 평가하고 모니터링하고자 할 수 있음
- 대략 이 비율은 10^{-3} 정도 될 것이 권장됨
- 이보다 낮으면 학습률이 너무 낮을 수 있음
- 이보다 높으면 학습률이 너무 높을 가능성이 있음

모니터링3: 업데이트 / 가중치값 비율

```
# assume parameter vector W and its gradient vector dW
param_scale = np.linalg.norm(W.ravel())
update = -learning_rate*dW # simple SGD update
update_scale = np.linalg.norm(update.ravel())
W += update # the actual update
print update_scale / param_scale # want ~1e-3
```