튜닝 프로세스

<체계적으로 하이퍼 파라미터를 튜닝할 수 있는 팁>

하이퍼 파라미터 종류 : 학습률, 모멘텀(adam 최적화 알고리즘의 하이퍼 파라미터), 베타원 베타투 앱실론, 층의 수, 층에서 은닉유닛의 숫자, 학습률 single 쓰지 않고 학습률 감쇠 쓸 것인지, 미니배치 사이즈

<파라미터의 중요도 순위>

1. 학습률 알파

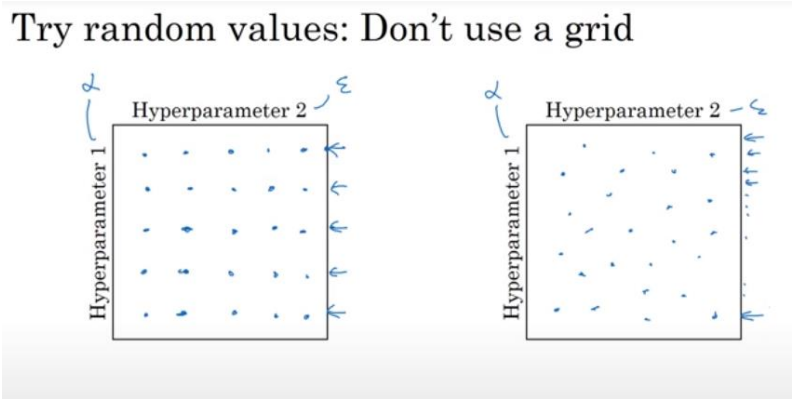
2. 모멘텀 (기본값 0.9) / 미니배치크기 튜닝 / 은닉 유닛의 수 튜닝

3. 층의 숫자 / 학습률 감쇠

아담 알고리즘에서는 베타원 베타투 앱실론은 튜닝을 하지 않고 0.9, 0.999, 10^(-8) 사용 (원한다면 튜닝해도 됨)

but 중요도가 딱 정해진 것은 아니고 다른 직관을 가질 수 있음

<만약 하이퍼 파라미터를 튜닝한다면 어떤 값을 탐색할지 어떻게 정할 수 있을까?>



머신러닝이 만들어진지 얼마 되지 않았을 때는

- 격자점을 탐색하는 것이 일반적이었다
- 예시에서 5x5니까 25개의 점만 생각하여 최고의 하이퍼 파라미터를 정함
- 하이퍼파라미터 수가 적을 때 쓸 수 있음

딥러닝에서는

- 무작위로 점을 선택
- 동일하게 25개의 점을 뽑는다 하면, 점에 대해 하이퍼파라미터를 정하는 것
- 왜냐면 어떤 하이퍼 파라미터가 문제 해결에 더 중요한지 미리 알 수 없고

하이퍼 파라미터의 중요도 순위가 있기 때문

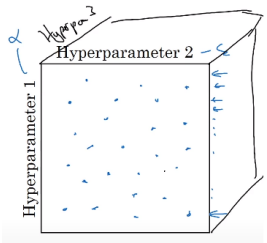
ex) 하이퍼 파라미터1이 알파, 하이퍼 파라미터 2가 앱실론(아담에서 분모 값) 이라고 하면

(알파를 고르는 것이 앱실론보다 중요)

- 5개의 알파값을 확인하게 되는데 앱실론이 달라도 결과는 같은 것을 확인할 수 있음
- 가장 중요한 하이퍼파라미터인 알파는 5개에 대해서만 학습시킨 것

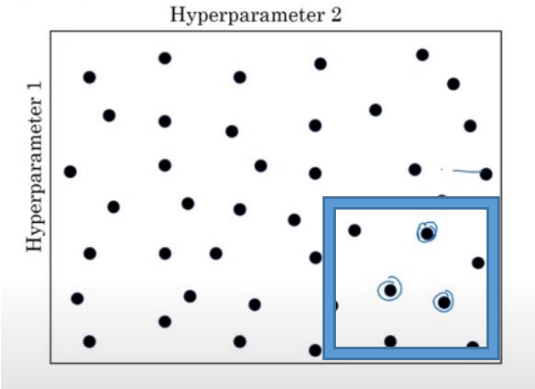
- 무작위로 모델을 고르면 25개의 서로 다른 알파를 이용하여 모델을 학습하여 더 좋은 하이퍼 파라미터 쓸 수 있음

<3개의 하이퍼 파라미터이면>



- 정육면체 안에서 모델을 고르는 것이어서 훨씬 많은 모델 학습하고 더 많은 하이퍼파라미터 탐색
- 어떤 하이퍼파라미터가 가장 중요한지 알 수 없으므로 무작위로 정하는게 더 하이퍼파라미터의 다양한 값을 탐색할 수 있음

<정밀화 접근>



- 2차원에서 이 점이 최고라는 것을 찾으면 그 주변도 괜찮은 성능
- 더 작은 영역으로 확대해서 더 조밀하게 점 선택
- 무작위인 것은 그대로, 최고의 하이퍼파라미터가 이 영역에 있다고 생각하여 파란색 안에 초점 두고 탐색
- 전체 사각형에서 탐색한 후 더 작은 사각형으로 범위를 좁혀 나감 더 조밀하게 시험함

★ 격자점이 아니라 무작위로 선택 + 원한다면 정밀화 접근을 이용할 수 있음

적절한 척도 선택하기

- 무작위라는 것이 가능한 값들 중 공평하게 뽑는 것이라고 할 수 없고 적절한 척도를 정하는 것이 중요

<가능한 값 중 무작위하게 뽑는 것이 합리적인 경우>

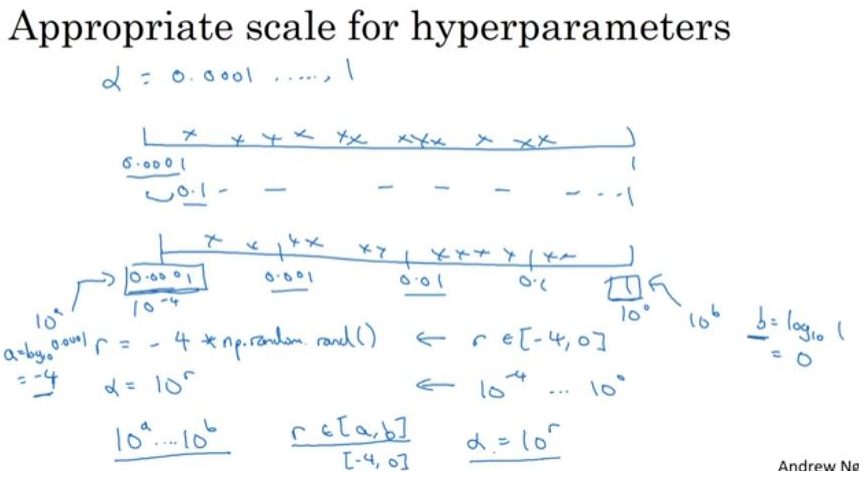
- layer l 에 대하여 은닉 유닛의 수 n^l 을 정할 때

  값의 범위로 50부터 100까지 의 수직선에서 무작위하게 값들을 고르는 것

- 신경망에서 레이어의 숫자 L을 구할 때

  2에서 4 사이에서 무작위하게 뽑고, 격자점을 사용해도 문제가 없음

<학습률 알파를 탐색하는데 00001부터 1까지를 생각할 때>



- 수직선 상에서 균일하게 무작위로 값을 고른다면 90프로의 샘플이 0.1과 1 사이에 있을 것
- 90프로를 0.1과 1 사이에 탐색하는데 쓰고 10프로만을 0.0001과 0.1 사이를 탐색하는데에 쓴다

  => 비합리적

- 선형척도 대신 **로그 척도**에서 하이퍼파라미터 찾는 것이 합리적!
- 로그 척도에서 균일하게 무작위로 값을 뽑는 것

  => 0.0001과 0.001 사이, 0.001과 0.01 사이를 탐색할 때 더 많은 자원을 쓸 수 있음

<파이썬 구현>

```
import numpy as np
r=-4*np.random.rand() #np.random.rand() : 0과 1 사이의 난수 생성
print(10**r) #[10^-4,1]
```

```
0.0023216616769839329
```

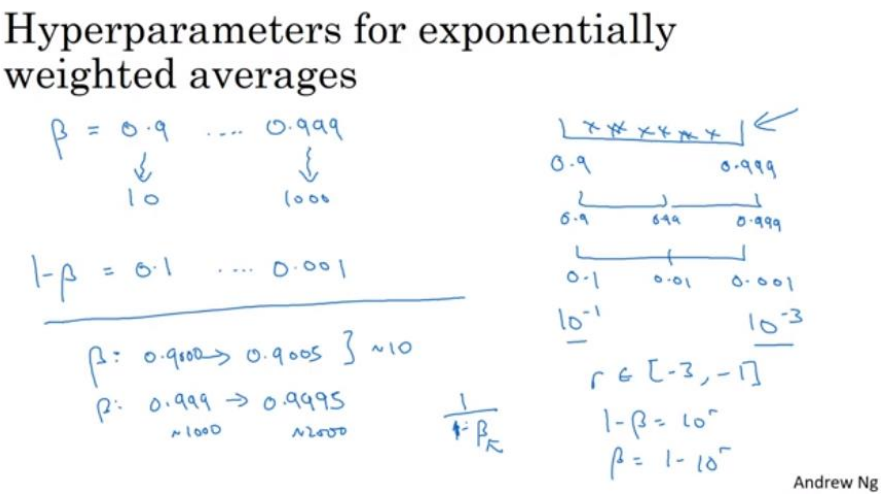
- 10^a 에서 10^b까지를 로그 척도로 탐색한다면
- 위의 예시에서는 10^a가 0.0001, a=-4 (a는 10을 밑으로 하는 로그를 취하면 구할 수 있음)
- 10^b=1, b=0

  ⇨ r은 a와 b사이에 랜덤하게 => 알파는 10^r

★ 낮은 값에서 로그를 취해서 a를 찾고, 높은 값에서 로그를 취해 b를 찾고 10^a 에서 10^b까지를 로그 척도로 탐색하는 것

★r은 a와 b사이에서 균일하게 무작위로 뽑으면 하이퍼 파라미터가 10^r

<지수가중 평균을 계산할 때 사용되는 하이퍼 파라미터 베타>



- 베타를 0.9와 0.999 사이에서 찾는다고 할 때 0.9일때는 지수가중 평균이 최근 10일의 평균 기온처럼 마지막 10개 값의 평균과 비슷하고 0.999의 경우에는 마지막 1000개 값의 평균과 비슷

  ⇨ 사이를 균일하게 무작위탐색하는 것은 합리적이지 않다

1-베타에 대해서 값 탐색하는 것이 합리적!

- 0.1, 0.01, 0.001 사이 탐색
- -3과1사이에서 균일하게 무작위로 값을 뽑는 것
- 1-베타를 10^r로 생각하면 되니 베타가 1-10^r
- 적절한 척도위에서 무작위로 하이퍼파라미터 샘플 추출함
- 이방법 이용하면 동일한 양의 자원 사용 0.9 0.99 와 0.99 0.999

<왜 선형 척도에서 샘플 뽑는 것 안좋을까?>

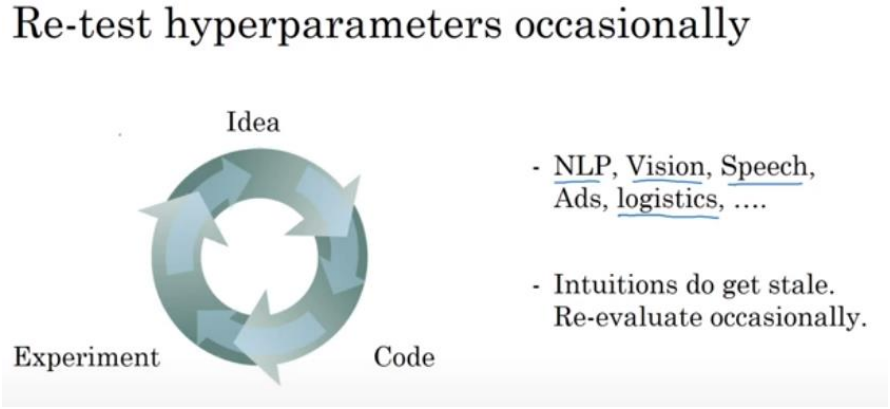
- 베타가 1에 가깝다면 베타가 조금만 바뀌어도 결과가 많이 바뀜
- 베타가 0.9에서 0.9005로 바뀌었다면 결과에 겨의 영향 주지 않음 => 대략 10개의 값 평균 내는 것
- 베타가 0.999에서 0.9995로 바뀌었다면 알고리즘의 결과에 큰 영향을 줄 것 => 마지막 1000개의 값의 지수가중평균을 내는 것에서 마지막 2000개 값의 평균을 내는 것으로 바뀌었기 때문
- 1/(1-베타) 가 베타가 1에 가까워질수록 작은 변화에도 민감하게 반응

  ⇨ 따라서 베타가 1보다 가까운 곳(1-베타는 0에 가까운 곳) 에서 더 조밀하게 샘플을 뽑음

- 하이퍼파라미터를 고를 때 적절한 척도 사용하지 않더라도 크게 걱정할 필요 없음

다른 척도가 우선하는 상황에서 균일한 척도에서 샘플링을 하더라도 정밀화 접근을 사용하면 괜찮은 결과 얻을 수 있고, 반복할수록 더 유의미한 하이퍼 파라미터 범위를 탐색하게 됨

하이퍼파라미터 튜닝 실전



- 한 어플케이션에서 얻은 하파에 대한 직관이 다른 영역에서 쓰일수도 아닐 수도 있음
- 서로 다른 어플 영역간에 공유되는 것

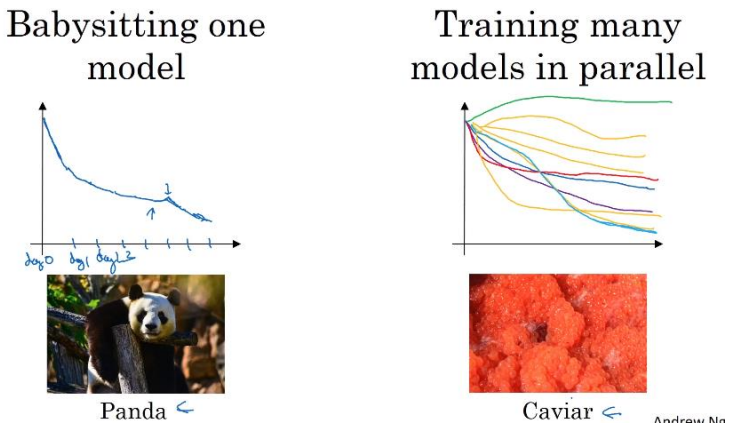
  ex) 컴퓨터비전 커뮤니티에서 발전된 컨프넷, 레스넷 : 음성, 자연어 처리에 적용

but 하이퍼파라미터를 찾는 과정은 그렇지 못한다는 직관

알고리즘을 발전시키거나 데이터가 바뀌거나 데이터 센터의 서버를 업그레이드 할 경우 하이퍼파라미터가 녹슴

⇨ 다시 시험해서 재평가해야함

<하이퍼 파라미터 찾는 방법>



1. 모델 돌보기

- 데이터는 방대하지만 CPU GPU 컴퓨터 자원이 많이 필요하지 않아서 적은 숫자의 모델을 한번에 학습시킬 수 있을 때

- 학습 과정에서 모델 돌보기

- 0일차에 무작위하게 매개변수 설정/학습

(학습 곡선에서 비용함수 J 나 개발 세트의 오차가 점진적으로 감소할 것)

- 1일차 끝 무렵=> 학습 속도 조금 올려서 더 나은지 확인

- 2일차=>모멘텀을 약간 올리거나 학습 속도 낮추기

이렇게 하이퍼파라미터 계속 조절하여 학습속도 조절..

매일 모델을 돌보며 학습시킴 / 성능을 잘 지켜보다가 학습 속도를 조금씩 바꾸는 방식

- 여러모델을 동시에 학습시킬 컴퓨터 자원이 충분하지 않을 때

- 판다 전략 (팬더는 한번에 한마리씩 아이 가지고 아기 팬더 살아남을 수 있도록 돌봄)

2. 여러 모델을 함께 학습

- 하파를 며칠에 걸쳐 스스로 학습하게 하고 동시에 다른 모델의 다른 하파 설정 다루기 시작

- 서로 다른 모델을 동시에 학습하고 여러 하이퍼파라미터 설정을 시험하여 마지막에는 최고 성능 뽑음

- 캐비어 전략 (물고기는 1억개의 알을 품고 하나에 집중하기 보다는 그 이상이 더 잘 살아남기를 지켜봄)

⇨ 두 접근 중 뭘 선택할지는 컴퓨터 자원의 양과 함수 관계

- 여러 모델을 동시에 학습시키기에 충분한 컴퓨터 => **캐비어 접근**

- 온라인 광고나 컴퓨터 비전 어플케이션 등 많은 데이터가 쓰이는 곳 / 모델이 너무 커서 한번에 여러 모델 학습 어려우면 => **팬더 접근**

⇨ 한 모델이 잘 작동하는지 확인한 이후 다른 모델 초기화하여 다시 돌볼 수도 있음