11주차

태그	완료

튜닝 프로세스

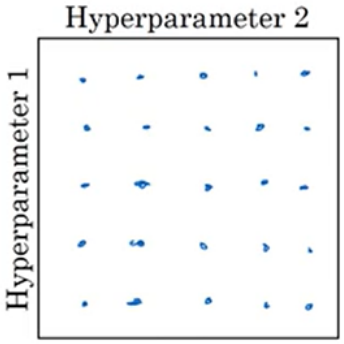
Hyperparameters

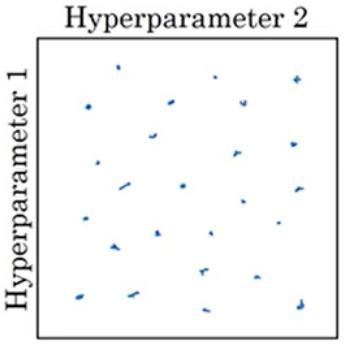
- 학습률 알파
- 모멘텀 최적화 기법: 베타
- Adam 최적화 기법: 베타1, 베타2, 엡실론 → 일반적으로 튜닝하지 않고 0.9, 0.999, 10^(-8)을 사용한다.
- the number of hidden layers
- the number of hidden units
- degree of learning rate decay
- mini-batch size

→ 빨강, 노랑, 초록 순서대로 중요하다.

Try random values: Don’t use a grid

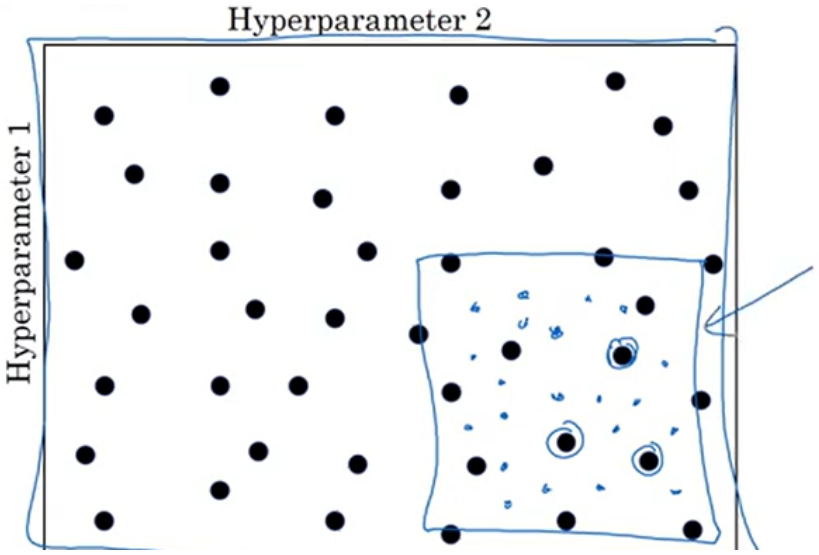
- hyperparameter 별로 중요도가 다르고 문제 해결에 어떤 hyperparameter가 중요한지 미리 알 수 없기 때문이다.
- 1번째는 grid, 2번째는 random이다.
- 예를 들어 Hyperparameter1 = 알파, Hyperparameter2 = 엡실론이라고 하자.
- grid에서는 엡실론보다 알파의 중요도가 훨씬 크기 때문에 같은 행에서는 엡실론이 어떻게 바뀌든지 같은 결과가 나올 것이다. 따라서 25개의 hyperparameter을 가정했지만 실질적으로는 5개의 알파에 대한 모델을 구한 것과 다름없다.
- 반면 random에서는 서로 다른 25개의 알파를 사용한 모델을 구할 수 있다.





Coarse to fine: 정밀화 접근

- 전체 hyperparameter 공간에서 탐색하여 좋은 파라미터가 있는 지역을 찾은 후, 그곳에서 집중적으로 hyperparameter를 탐한다.



적절한 척도 선택하기

- random하게 hyperparameter를 고르는 것이 반드시 균일 분포를 따른다고 볼 수 없다 → 대신 적절한 척도를 정하는 것이 더 중요하다.

Picking hyperparameters at random

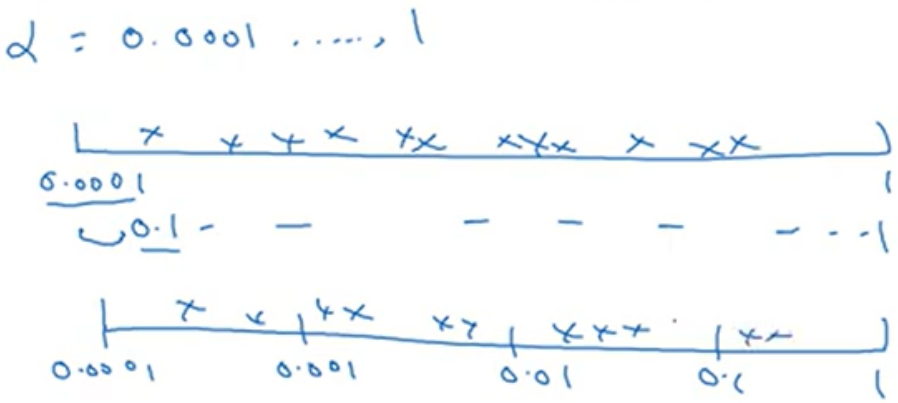
- hidden layer L의 hidden unit 개수 $n^{[L]}$를 50~100 사이에서 정하거나 # layers를 2~4 사이에서 정하는 것은 random으로 뽑는 것이 합리적인 경우다.

Appropriate scale for hyperparameters

- 그러나 학습율 알파를 0.0001~1 사이의 값에서 정할 때, 90%의 샘플이 0.1~1 사이에 있을 것이고 10%의 샘플이 0.0001~0.1 사이에 있을 것이다. 따라서 0.0001~0.1 사이

의 값을 제대로 탐색하지 못하게 된다. → 비합리적

⇒ 적절한 scale을 사용: 선형 척도 대신 로그 척도에서 hyperparameter를 탐색하면 범위 내에서 고르게 탐색 가능하다.



```
# 예시 1
r = -4 * np.random.rand() # r은 [-4, 0] 사이의 값
알파 = 10^r # 알파는 [10^(-4), 1] 사이의 값

# 예시 2
10^a = 0.0001 = 10^(-4) -> a = -4
10^b = 1 = 10^0 -> b = 0
```

- 로그 척도로 탐색하는 법
  1. 낮은 값에 log를 취해 a를 찾고 높은 값에 log를 취해 b를 찾는다.
  2. r의 범위는 [a, b]이고 이 범위에서 샘플을 random하게 탐색하면 10^r이 hyperparameter가 된다.

Hyperparameters for exponentially weighted averages

- 예를 들어 지수가중평균에 쓰이는 베타를 0.9~0.999 사이의 값에서 찾는다고 하자.
- 베타 = 0.9라면 지수가중평균이 10일 동안의 기온 평균과 같고 베타 = 0.999라면 지수가중평균이 1000일 동안의 기온 평균과 같다. (days = $\frac{1}{1-\beta}$)
- 이때 0.9~0.999사이를 선형 척도로 random sampling하는 것은 비합리적 → (1 - 베타)를 이용해 random sampling한다.
- (1 - 베타)는 0.001~0.1 사이의 값으로, r은 [-3, -1]의 범위를 갖게 된다. (1 - 베타) = 10^r이므로 베타 = 1 - 10^r이다.

- 이렇게 하면 0.9~0.99, 0.99~0.999 사이의 값들을 균일하게 탐색할 수 있다.
- 그렇다면, 왜 선형을 척도로 쓰는 것이 안 좋을까?
  - 베타가 1에 가까울 때, 베타의 값이 조금만 변해도 결과가 크게 바뀌기 때문이다.
  - 예를 들어 베타 = 0.9 → 베타 = 0.9005로 값이 바뀌어도 두 경우 모두 대략 10개의 샘플을 탐색한다. 그러나 베타 = 0.999 → 베타 = 0.9995로 값이 바뀌면 1000개의 샘플에서 2000개의 샘플을 탐색하는 것으로 크게 바뀐다. ⇒ 베타가 1에 가까운 곳에서 샘플을 더 조밀하게 뽑는다. (비합리적인 이유)

하이퍼파라미터 튜닝 실전

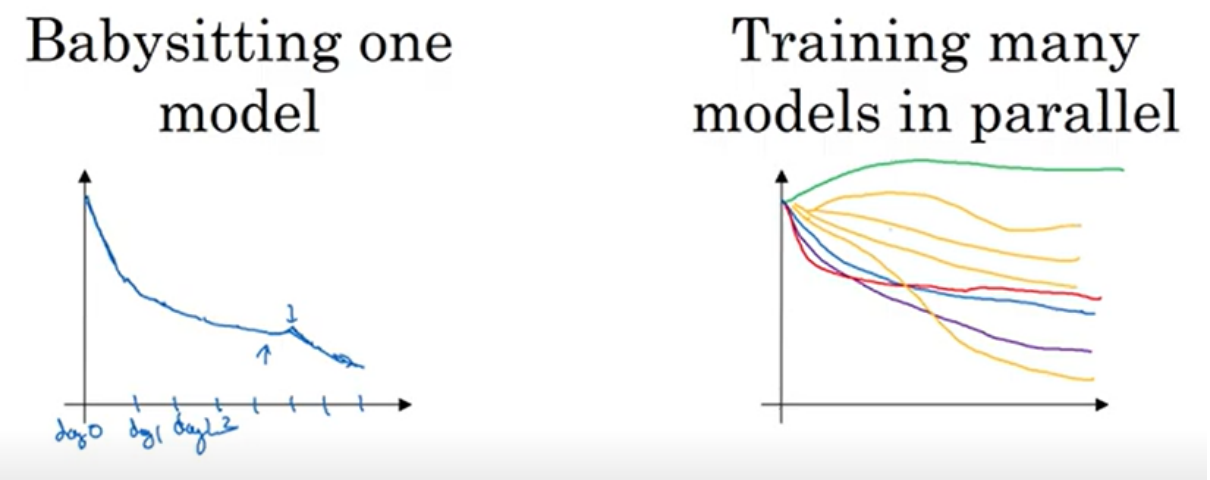
- 여러 domain: NLP, CV, Speech...
- 한 domain에서 얻은 hyperparameter에 대한 직관이, 다른 domain에서는 잘 적용되지 않을 수 있다.

Babysitting one model: Panda Approach

- 데이터는 방대하지만 컴퓨터 자원이 충분치 않아 한 번에 여러 모델을 학습시키기 어려운 경우
- 몇 주에 걸쳐 매일 하나의 모델을 돌보며 학습시키는 것으로, hyperparameter를 하나의 모델에 대해 계속 다르게 시도한다.

Training many models in parallel: Caviar Approach

- 컴퓨터 자원이 충분한 경우
- 서로 다른 hyperparameter를 가진 여러 개의 모델을 동시에 학습시키고 가장 좋은 곡선을 찾는다.



배치 정규화

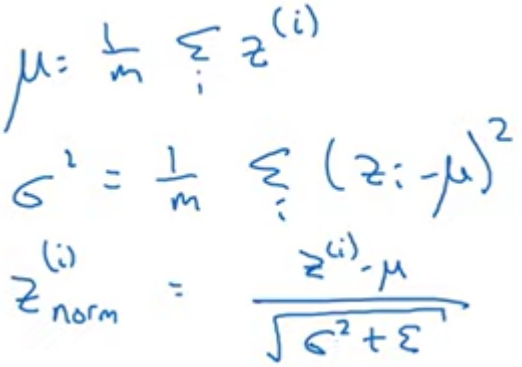
- 배치 정규화는 hyperparameter의 탐색을 쉽게 만들어주고 신경망과 hyperparameter의 상관관계를 줄여준다. 또한 더 많은 hyperparameter가 잘 작동할 수 있도록 돕고 깊은 신경망에서 학습이 잘 되도록 돕는다.

Noramalizing inputs to speed up learning

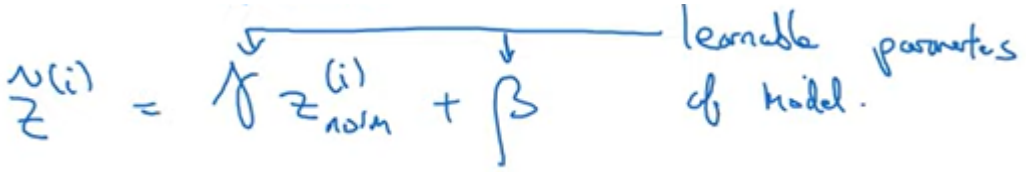
- hidden layer의 $w^{[l]}, b^{[l]}$를 빠르게 학습시킬 수 있도록 input인 활성화 값 $a^{[l]}$을 정규화시킬 수 있는가? → normalize $z^{[l]}$

Implementing Batch Norm

- l번째 layer의 node들의 z값: $z^{(1)} \dots z^{(m)}$: 일반적으로 normalization을 통해 z들의 평균이 0, 표준편차가 1이 되도록 만듦 → 그러나 hidden unit의 값들은 다양한 분포를 가져야 하므로 항상 이런 분포를 갖도록 하는 것은 좋지 않다.



- 따라서 정규화된 z를 감마와 베타를 이용해 다시 선형변환해 줄 수 있다. 감마와 베타는 learnable parameter로, 감마와 베타를 이용하면 hidden unit의 입력값, z가 서로 다른 평균과 표준편차를 갖도록 할 수 있다. → use z^~ instead of z
  - 만약 감마 = $(\sigma^2 + \varepsilon)^{0.5}$, 베타 = μ라면 z^~ = z이다.

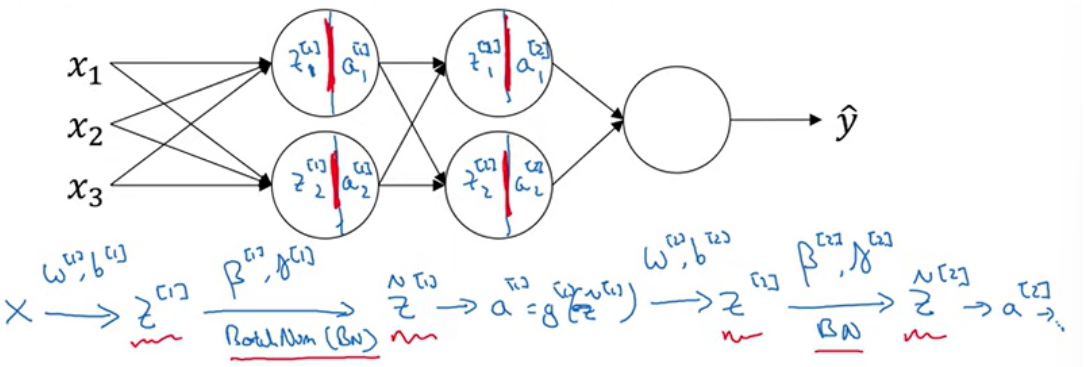


> 💡 **배치 정규화의 key idea**
>
> -입력값 X뿐 아니라 hidden unit의 입력값 z도 normalize하자!
> -이때 감마와 베타를 이용해 z가 다양한 평균과 분산을 갖도록 하자!

배치 정규화 적용시키기

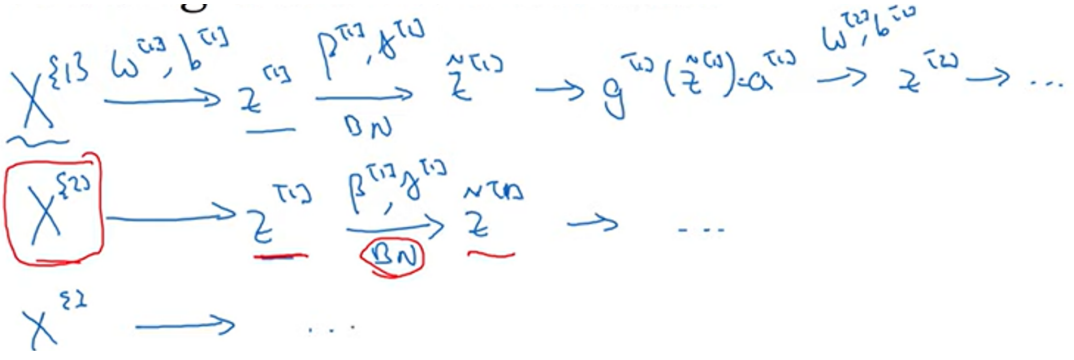
Adding Batch Norm to a network

- 배치 정규화가 z의 계산과 a의 계산 사이에 이루어진다.
- 이때 베타는 최적화 알고리즘에서 쓰이는 베타와 구별된다.
- tf.nn.batch-normalization

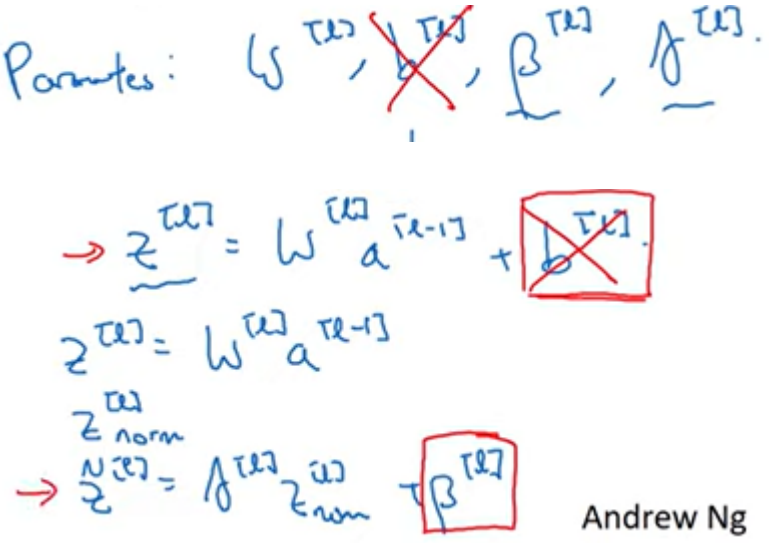


Working with mini-batches

- 각각의 미니 배치에 대해 배치 정규화를 진행하며 z와 a를 계산한다.
- 이때 배치 정규화를 위한 평균과 분산을 계산할 때는 그 미니 배치에 있는 데이터만을 사용한다.



- 한 번의 배치 정규화에는 네 개의 파라미터가 쓰인다. 그러나 $z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$ 에서 b는 z에서 평균을 빼주는 과정에서 사라지게 된다.
- $z^{[l]}$이 ($n^{[l]}$, 1) 이고 감마와 베타는 z를 scaling 해주기 때문에 모두 ($n^{[l]}$, 1) 차원을 갖는다.



Implementing gradient descent

```
for t = 1...num_of_mini_batches
    compute forward propagation of X^{t}
        in each hidden layer, use Batch Normalization to
        replace z^l with z^~l

    compute backpropagation to compute dW, db, dbeta, dgamma
    update parameters
    # gradient descent
    W:= W - alpha*dW, beta:= beta - alpha*dbeta,
    gamma:= gamma - alpha*dgamma

    # also works with momentum, RMSProp, Adam
```

배치 정규화가 잘 작동하는 이유는 무엇일까?

1. 서로 다른 scale을 갖는 입력값 X에 대해 비슷한 범위를 갖도록 정규화하여 학습 속도를 높인다. 이 작업을 입력값과 hidden unit의 값에 모두 적용한다.
2. learning on shifting input distribution

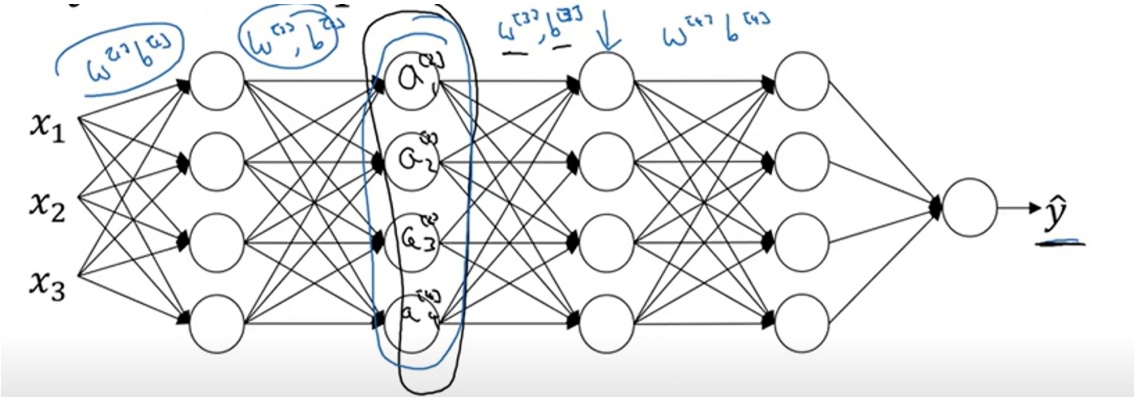
a. 배치 정규화는 뒤쪽 층이 이전 층의 가중치 영향을 덜 받게 한다.

b. 예를 들어 고양이 사진을 분류하는 모델에서 검정색 고양이만으로 모델을 훈련시켰다고 한다면, 다른 색의 고양이는 잘 분류하지 못할 것이다. 이처럼 X의 분포가 바뀌었을 때 모델은 잘 동작하지 못한다. → covariance shift 문제

c. covariance shift 문제: 만약 아래 그림에서 1번째 층을 제하고 생각해보면, 3번째 층의 입력값은 a^[2]_1, a^[2]_2, a^[2]_3, a^[2]_4가 될 것이고, 이에 따라 3, 4번째 층의 w, b가 결정될 것이다. 그러나 만약 1번째 층의 w, b가 변한다면 2번째 층의 활성값이 변할 것이고, 이에 따라 3, 4번째 층의 w, b도 변할 것이다.

d. 이때 배치 정규화는 앞쪽 층의 w, b가 바뀌어도 뒤쪽 층의 입력으로 들어가는 z들의 분포가 변화하는 양을 줄여준다(분포가 일정하도록 제한한다). → z값들이 바뀌더라도 z의 평균과 분산은 일정하게 유지된다.

e. 앞쪽 층이 변화해도 뒤쪽 층을 쉽고 안정적으로 학습할 수 있다. 또한 앞쪽 층의 w, b와 뒤쪽 층의 w, b간의 관계를 약화시킨다 → 다른 층에 상관 없이 각 층의 w, b이 스스로 학습할 수 있다.



3. Batch Norm as regularization

a. 미니 배치 X^{t}의 z^[l]에 대해 그 미니 배치의 데이터만을 이용해 해당 미니 배치 평균과 분산을 구함 → 전체 데이터의 평균 / 분산보다 더 큰 잡음을 갖고 있다.

b. noise가 끼어있는 평균 / 분산으로 계산하기 때문에 z^[l] → z^~[l]으로 변환하는 과정에도 noise가 발생한다.

c. 즉, dropout처럼 hidden layer의 activation function에 noise가 끼어 있다. (dropout에서는 drop하면 0을 곱하고 keep할 거면 1을 곱하기 때문에 곱셈 잡음이 끼어있다고 할 수 있다) → 배치 정규화는 평균으로 빼니 덧셈 잡음과 표준편차로 나누니 곱셈 잡음이 있다. 이때 평균과 표준편차 자체에도 잡음이 있다.

d. 앞선 hidden layer에 잡음이 끼어 있으면 이후의 은닉층이 하나의 hidden layer에만 의존하지 않는다.

e. 따라서, 배치 정규화는 dropout처럼 약간의 일반화 효과를 갖는다.

f. 그러나, 더 큰 사이즈의 미니 배치를 사용할수록 잡음이 줄어드므로 일반화 효과가 약해진다. 따라서 배치 정규화를 일반화 목적으로 사용하지 않는다.

테스트시의 배치 정규화

- 배치 정규화는 한 번에 하나의 미니 배치 데이터를 처리한다. 그러나 테스트 시에는 미니 배치가 없기 때문에 하나의 데이터씩 처리한다.
- m은 전체 데이터 개수가 아니라 하나의 미니 배치에 있는 데이터 수이다. 엡실론은 σ^2이 너무 작을 때를 대비한 수학적 수이다.

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)}-\mu)^2$$

$$z^{(i)}_{\text{norm}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

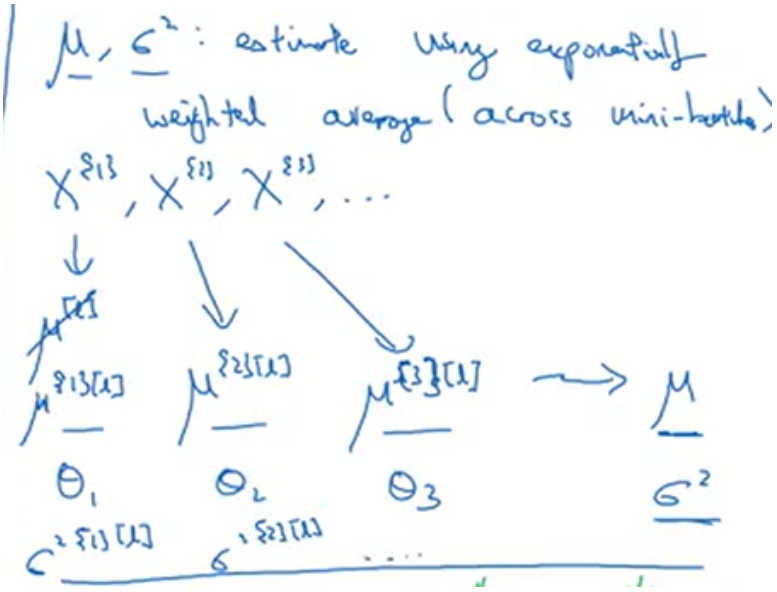
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$$

→ 그러나 모델을 테스트할 때는 미니 배치 개념이 없고, 데이터 한 개의 평균과 분산을 구하는 것은 말이 안 되므로 별도의 방법이 필요하다.

Batch Norm at test time

- 각 독립된 μ와 σ^2를 사용하면 된다.
- 테스트에 사용되는 평균과 분산은 train set으로부터 얻어야 한다.
  - 일반적으로 배치 정규화를 구현할 때 사용하는 평균과 분산은 미니배치의 지수가중평균으로 구한다.
  - hidden layer l에 입력으로 미니배치 X^{1}, X^{2}...이 들어간다고 하자.
  - 각각의 미니배치로부터 l층의 평균$(\mu^{1[l]}, \mu^{2[l]} \ldots)$과 분산$(\sigma^{1[l]}, \sigma^{2[l]} \ldots)$을 구할 수 있다.

- 이때 각각의 미니배치에 대응하는 θ_i(i = 1...k)가 있다고 하면 이를 이용해 평균 μ와 분산 σ^2의 지수가중평균을 구할 수 있고, 이것이 각 층에서의 z값의 평균과 분산의 추정치가 된다.



Quiz

✓ 3. β(모멘텀의 하이퍼파라미터)가 0.9에서 0.99 사이의 값일 때, 아래의 코드에 들어갈 알맞은 것을 선택해 주세요. *1/1

```
r = np.(a)
beta = 1 - (b)** (-r - 1)
```

- (a) - random.randn(), (b) - 100
- (a) - random.randn(), (b) - 10
- (a) - random.rand(), (b) - 100
- (a) - random.rand(), (b) - 10 ✓

- (a): np.random.rand()는 균일 분포를 따르는 [0, 1) 사이의 값을 랜덤하게 추출, np.random.randn()은 평균 0, 표준편차 1인 표준정규분포를 따르는 값을 랜덤하게 추출(음수가 나올 수 있음)

- (b): 베타가 0.9~0.99이므로 1 - 베타 = 10^r은 0.01~0.1사이다. 따라서 r은 [-2, -1]이다. 그러나 이 문제에서 r은 [0, 1)이므로 범위를 맞춰주려면 베타 = 1 - 10^(-r-1)을 해야 한다.