**Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization**

|  |  |  |
| --- | --- | --- |
| Training set | Hold-out cross validation | Test set |

훈련 다양한 모델 중 성능이 좋은 모델 확인 최종모델 평가

이전 - 70% train : 30% test  
현재 – 60% train : 20% validation : 20% test (상대적으로 작은 데이터셋 일 때)

테스트셋 : 편견 없는 추정치 제공 -> 필요하지 않다면 없어도 됨(val set에서 평가)

[ Bias and Variance ]

high bias – underfitting  
high variance – overfitting

고차원에서 편향과 분산 시각화 불가능 -> 다른 메트릭 ; train set error / dev set error  
train 성능 >> dev 성능 : 훈련셋 과적합, 검증셋 일반화 X => high variance  
train 성능 < dev 성능 : 과소적합 => high bias

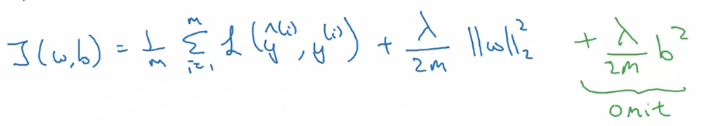
High Bias가 있는지 확인 -> 훈련 세트나 훈련데이터 성능 살펴봐야  
있으면, 훈련 세트에 잘 맞지 않거나 더 오래 훈련할 수 있음  
=> 네트워크 선택시도 (더 많은 은닉계층, 은닉유닛, 오랜 훈련, 최적화 알고리즘)

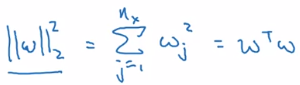
Bias를 줄이면 평가하기 위해 검증세트 성능 살펴봄(분산문제)  
-> 분산 문제 있으면, 더 많은 데이터를 얻으면 좋거나 정규화를 해야 한다.

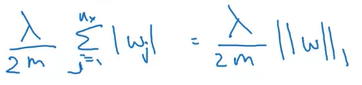
머신러닝 초기엔 bias-variance trade-off 문제  
-> 현재 딥러닝 빅데이터시대로 더 큰 네트워크, 더 많은 데이터 얻으며 문제 감소  
(정규화로 분산 줄어들지만, 약간의 trade-off 문제 생길 수 있음)

[L2 정규화] – 과적합 방지, 고분산 해결

로지스틱 회귀 손실함수

  
 규제항 : w제곱의 노름 매개변수 b는 생략

 벡터 w에 대한 제곱 유클리드 노름 => L2 정규화  
 => 가중치 감소

매개변수 w는 벡터이며 고분산 문제가 있다.(많은 변수)  
매개변수 b는 매우 많은 수의 매개변수(w)에 대한 하나의 매개변수이기에 생략 가능 L1 정규화 => w는 희박해져 많은 0을 가짐

매개변수 집합이 0이면 모델을 저장하는데 더 적은 메모리가 필요하기에 모델을 압축시킴

람다 : 정규화 파라미터(하이퍼파라미터) - python에서는 lambd로 사용  
노름 : Frobenius norm

IF 람다를 크게 하면, w를 0에 가깝게 설정하도록 유도 (손실함수 최소화를 위해)  
 => 은닉 유닛의 많은 영향을 없앰 -> 작은 신경망이 됨(단순한 네트워크) <정규화 효과>  
 => 하지만, 여러 층을 깊이 쌓아야 하는데, 이런 과적합한 경우에서(고분산) 큰 편향을 갖도록

w작으면, z도 작아짐 -> z는 작은 범위이므로, g(z)는 대략적으로 선형일 것이다. (g(z)=tanh)  
=> 모든 층이 선형이면, 전체 네트워크가 선형 네트워크 => 과적합 줄어들음

[드롭아웃 정규화]

과적합일 때, 드롭아웃을 진행해 신경망의 노드를 제거하는 확률 세팅 => 간단한 네트워크

<Inverted Dropout>

l = 3  
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep-prob   
# (keep-prob = 0.8 : 특정 은닉층이 유지될 확률 / 0.2 : 은닉층 제거할 확률)

a3 = np.multiply(a3, d3) # a3 \* d3 (element-wise) : 각 요소끼리 연산  
=> d3의 요소들을 0으로 만드는 역할 (0(False)과 1(True)로 만들어주는 boolean array)

a3 /= keep-prob # inverted dropout technique

세번째 층에 50 단위(개)의 신경세포가 있다.   
=> a3 : 50 \* m 차원 => (20%) 10 유닛은 닫거나 0으로 됨  
=> z[4] = w[4] \* a[3] + b[4] -> 20%의 a3의 element가 0이 됨  
=> z[4]의 기대값을 감소하지 않게 하기 위해서는 a[3] /= 0.8 (다시 20%만큼 올라갈 수 있기에)  
=> a[3]의 기대값 변하지 않음

inverted dropout technique : keep-prob를 어떻게 설정하더라도 a[3]의 기대값 동일하게 유지

신경망을 평가할 때, inverted dropout technique이 test time을 쉽게(간단하게) 만들어준다. 스케일링 문제가 덜하기 때문에

d vector는 첫 번째 기울기 강하 수행절차에서, 동일한 훈련셋에서 multiple pass를 진행하면, 훈련셋에 통과하는 다른 pass들에 따라 다른 은닉층을 무작위로 0으로 만든다.   
=> 두 번째 기울기 강하에서 훈련셋을 2번째로 통과시켜 또 다른 패턴이 은닉층을 0으로 만들어야 하는 것은 아니다.  
=> 세 번째 층의 d vector는 어떤 것을 0으로 만들지 결정하는데 사용된다.

노드가 임의로 제거될 수 있기에 다음 노드는 어떤 한 가지의 특성에 의존하면 안된다.  
입력 노드 전체에 비중을 나눠 가중치 분산시키면 가중치의 squared 노름을 줄이는 효과가 있고, L2정규화처럼 드롭아웃 구현의 효과는 강도가 비슷하며 과적합을 방지하는 데 도움된다.  
=> 요약) 드롭아웃이 L2정규화와 유사한 효과 가짐 (적용되는 부분에서의 차이만 있음)

w[l]의 shape이 클수록, 매개변수 세트가 크기에 가중치가 가장 큰 행렬이다. 그 행렬의 과적합을 줄이기 위해 keep-prob가 비교적 낮을 수 있다 (낮을수록 출력이 0이 되는 element 많다)  
\* keep-prob = 1.0 : 모든 유닛 유지, 해당 레이어에 드롭아웃 사용하지 않음

Computer Vision(CV)에는 드롭아웃이 흔히 사용됨

드롭아웃 단점 : 비용함수가 모든 반복에서 더 이상 잘 정의되지 않음

[조기종료]

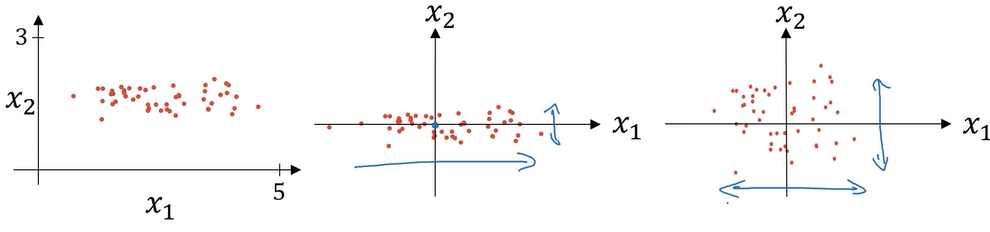
train error / 비용함수 : 항상 감소  
dev set error : 감소하다 어느 순간 증가 => 그 최저점에서 조기종료

w의 비율이 중간정도 되는 시점에서 중지  
L2 정규화와 비슷하게, 신경망에서 w 파라미터 수가 비슷한 노름을 선정해 덜 과적합되게 만들어주는 것

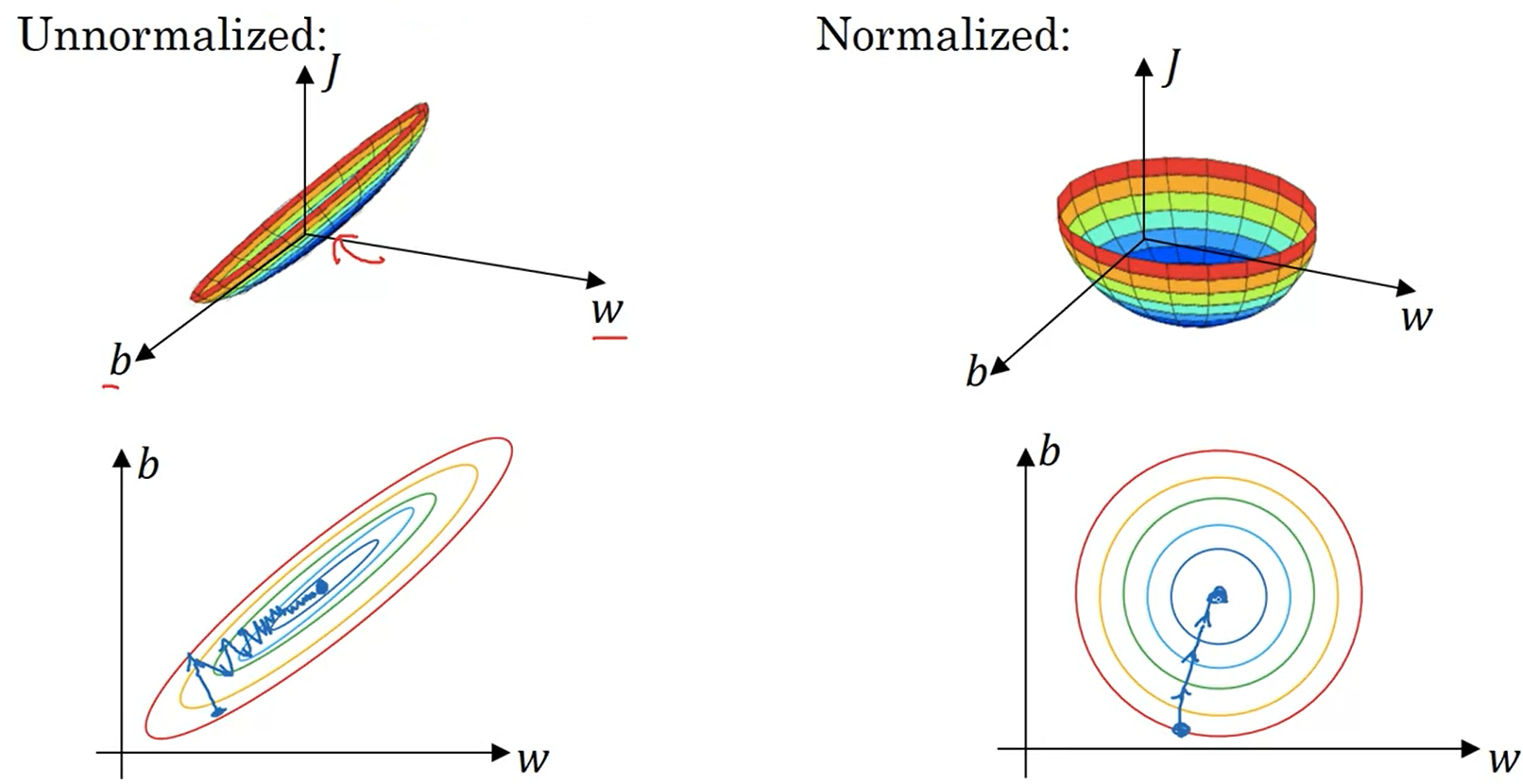
단점 : 이 두 별개 문제를 단독으로 풀 수 없고 묶는다. => L2 정규화 사용이 더 좋음  
- 비용함수가 최저가 되는 파라미터 w, b 찾기   
- overfit이 되지 않게 하는 정규화나 데이터 수 늘리기

장점 : 기울기 강하를 한 번만 실행해도 작은 w값, 중간값, 큰 값을 한번에 구할 수 있다.

[입력을 정규화] – 신경망 훈련 속도 높임



* 평균을 빼거나 0으로 만드는 단계
* 분산을 정규화하는 단계



[기울기 폭발/소실 문제] – 역전파과정에서 입력층에 가까워질수록 gradient 값이 매우 커지는 / 작아지는 현상 => 활성화함수 기울기 원인 \* Xavier 초기화

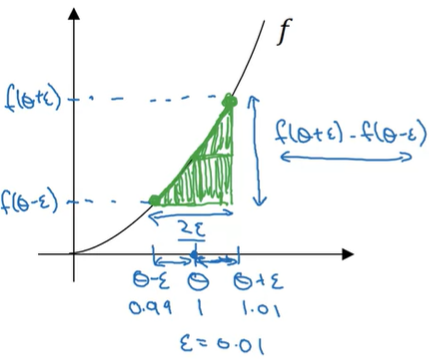
w[l] > I(단위행렬) : 활성화가 기하급수적으로 증가  
w[l] < I : 활성화가 기하급수적으로 감소

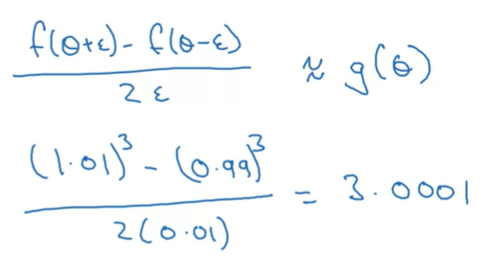
* 신경망의 무작위 초기화

z = w1x1 + w2x2 + w3x3 + … + wnxn

z가 너무 작지 않게 하기 위해 n(입력특성 개수)이 클수록 w\_i가 작기를 바람(각각의 항이 작아야)  
Var(wi) = 1/n (Relu일 경우, 2/n tanh이면 1/n)  
w[l] = np.random.randn( , ) \* np.sqrt(1/n[l-1]) (Relu일 경우, 2/n[l-1])

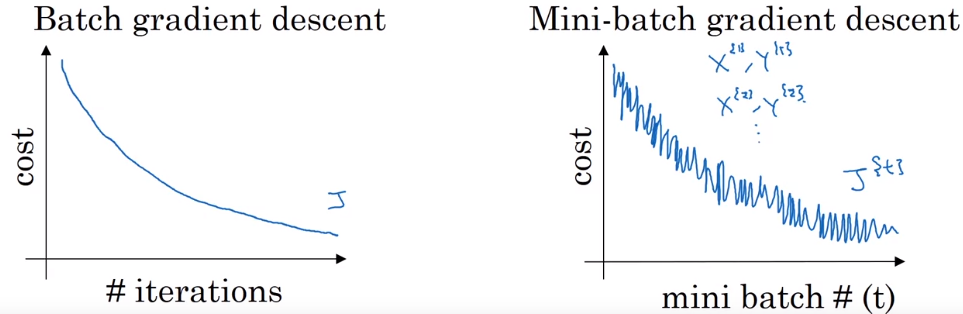
[two-sided difference]

f(세타) = 세타 ^3



[최적화 알고리즘]

mini-batch - 대규모에서 빠름

m개의 샘플 ; X{1}, X{2}, … X{m/1000} => 하나의 미니배치는 1000개 샘플 (X{1})  
X : (n\_x, m) X{1} : (n\_x, 1000) m/1000번 -> 1epoch  


mini-batch size = m : Batch gradient descent (X{1}, Y{1}) = (X, Y)  
- 반복마다 거대한 훈련셋 처리  
- 반복마다 긴 시간 소요 (큰 데이터셋일 때)

* 미니배치를 사용한 1 epoch 속도 >배치를 이용한 1 epoch 속도 => 항상은 아님

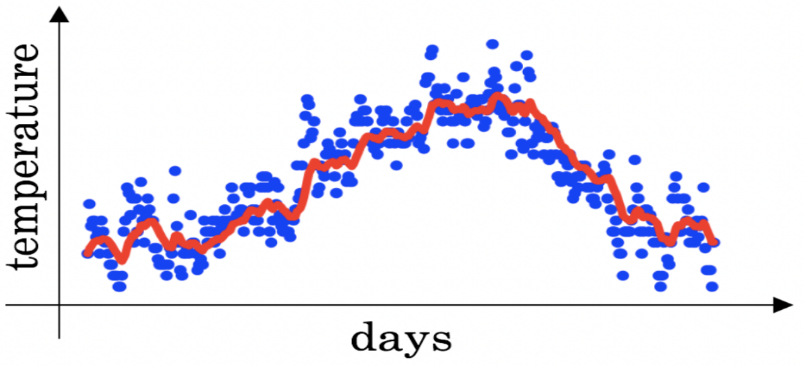
mini-batch size = 1 : Stochastic gradient descent (확률적 경사 하강법)  
- 노이즈는 더 작은 값의 학습속도를 사용해 개선시키거나 감소시킬 수 있음  
- 벡터화에서 속도를 잃음 (1개이기에)

<mini-batch size 고르기>

작은 훈련셋 (m <= 2000) -> 배치 경사 하강법

보통 -> 64, 128, 256, 512

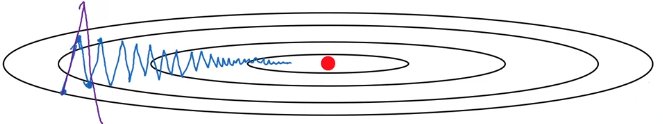
< 지수 가중 평균 >  
 V\_0 = 0 (초기화)

B = 0.9 -> red line

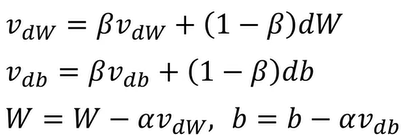
B 증가  
-> red line이 약간 오른쪽으로 이동

B 감소  
-> red line 안에서 진동 심해짐

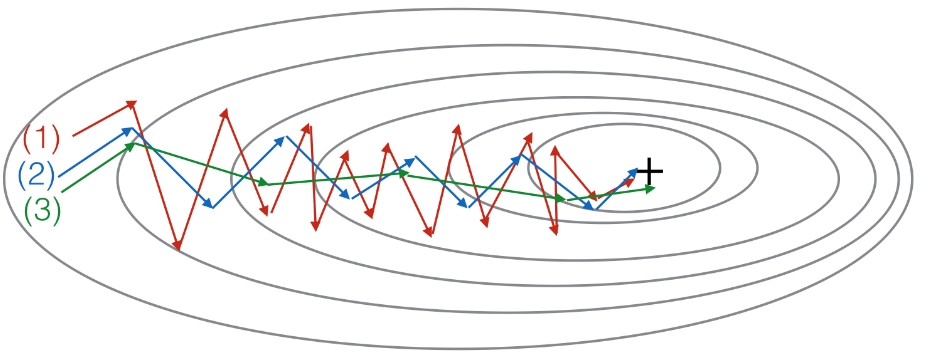
[모멘텀 기울기 하강] - 기울기의 지수 가중평균 산출해 가중을 업데이트 하는 것 / 이동평균사용



세로축에서 학습률이 작기를 바람(진동이 커짐을 방지하기 위해) / 가로축에서는 빠른 러닝 원함  
변동의 기울기의 평균화를 구하면, 세로축의 방향의 변동은 거의 0에 가까운 값으로 평균화됨  
가로축 방향은 모든 미분이 가로 방향의 오른쪽을 가리켜 평균이 꽤 크기에 빠르게 움직임  
미니 배치가 전체 훈련셋인 경우, 즉 배치 기울기 하강일 경우 잘 작동해 dw(큼), db(작음) 계산

dW, db : 경사 아래로 내려가고 있는 악셀 역할  
V\_dw, V\_db (모멘텀항) : 속도   
B(<1) : 마찰력 역할, 제한없이  
 속도가 붙는 것을 막아줌

하이퍼파라미터 alpha(고정) ,Beta(0.9) : 지수 가중 평균 조정



1. : gradient descent
2. : gradient descent with momentum (small B)
3. : gradient descent with momentum (large B)

[RMS prop] (Root Mean Square)

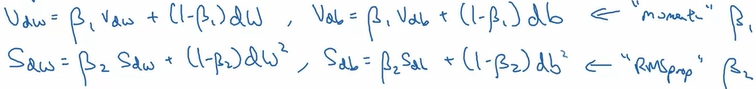
S\_dw = B \* S\_dw + (1-B) \* dW^2  
S\_db = B \* S\_db + (1-B) \* db^2

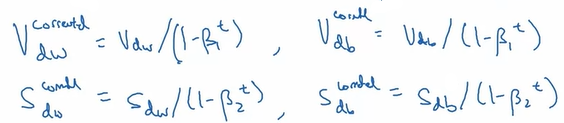
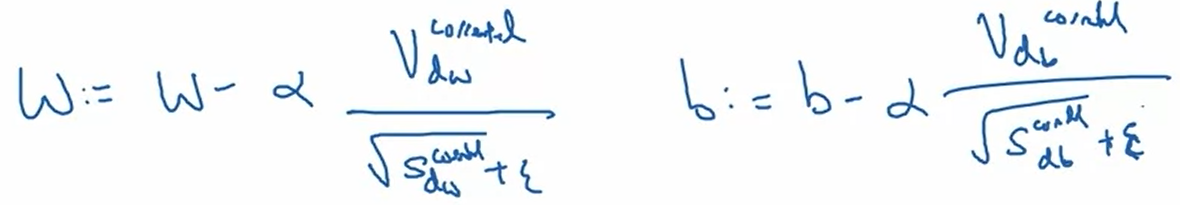
W방향(가로방향) – 빠른 학습  
b방향(세로방향) – 느린 학습 원함 (느린 학습으로 진동 큼을 방지, 기울기 클수록 빠른 학습)

가로방향의 변동은 증가시키고, 세로방향의 변동은 늦추고 싶기에 S\_dw는 상대적으로 작고, S\_db는 상대적으로 큰 값이 되어야 한다.  
=> 그래야 W의 변화량이 커지고, b의 변화량이 작아진다.  
=> S\_db가 큰 값이므로, 기울기는 b방향으로 더 크게 된다.

진동을 감소시키는 효과가 있고, 감소시키면 더 큰 learning rate를 사용할 수 있고, 속도 증가됨

[Adam] – RMSprop + 모멘텀 => 적응적인 모멘트 추정

V\_dw = V\_db = S\_dw = S\_db = 0  


Adam은 바이어스 수정도 같이 함  
- 바이어스 수정 후의 평균 (Beta의 제곱 취해줌)  
  


<하이퍼파라미터>

alpha(learning\_rate) : 조정  
Beta1(가중평균 dw / 모멘텀과 같은 항) : 0.9 -> 미분의 평균값 (1차 모멘트)  
Beta2(이동 가중 평균 dw^2) : 0.999 -> 제곱의 지수 가중 평균 계산 (2차 모멘트)

[학습률 감소]

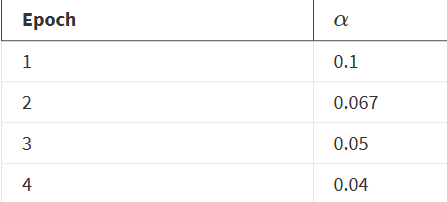
: 학습 알고리즘 속도를 높이기 위해 시간이 지남에 따라 학습률을 천천히 줄이는 것

학습률 고정 + 잡음 -> 최저점에 수렴 못할 가능성 있음, 주변 배회  
학습률 큰 동안 비교적 빠른 학습, 학습률 작으면 최소의 좁은 지역에서 진동

1 epoch = 데이터를 한 번 통과

decay-rate : 학습률 감소율

alpha0 : 초기 학습률

ex ) 초기 학습률 = 0.2   
/ 학습률 감소율 = 1

학습률 점차 감소

<지수 감소>

alpha = 0.95^epoch\_num \* alpha0 : 학습률을 기하급수적으로 떨어트림

alpha = (k / sqrt(epoch\_num)) \* alpha0

alpha = (k / sqrt(t)) \* alpha0 (t : 미니배치 수)

[로컬 옵티마]

안장점(saddle point) : 비용 함수에서 기울기가 0인 지점

plateaus : 실제로 학습속도를 늦출 수 있으며, 미분이 오랫동안 0에 가까운 영역  
-> 이 구간을 빠져나오기 위한 시간이 오래 걸림 => 학습을 늦추게 하는 요소 => 문제 !!

비교적 신경망이 큰 네트워크를 훈련하는 이상, 안좋은 local optima에 갇힐 확률은 작으며 비용함수는 비교적 고차원의 공간에서 정의된다. 또한 plateaus가 문제가 돼서 학습의 속도를 저하할 수 있는데, 모멘텀 / RMSprop / Adam과 같은 알고리즘이 plateaus를 빠져나오는 속도를 높인다.

[하이퍼 파라미터 튜닝]

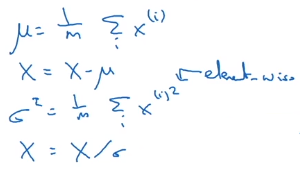
<중요도>  
학습률 alpha > 모멘텀 Beta(0.9) / 은닉층 개수 / 미니배치 size  
 > 층 개수 / 학습률 감소 > Adam Beta1(0.9) , Beta2(0.999), epsilon(10^-8)

<집합> - 그리드서치 사용하지 않고, 무작위로 추출하는 것이 더 좋다.

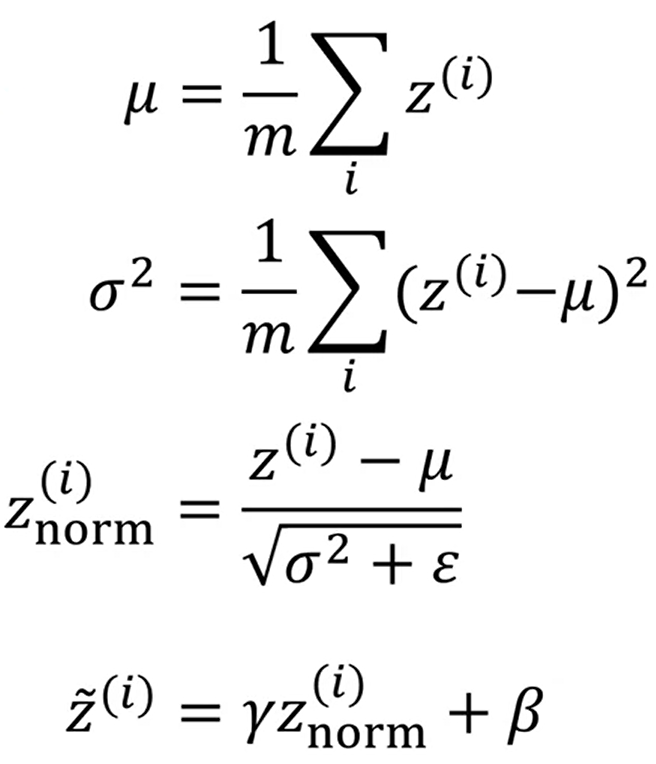
* 특정 범위의 유효한 값에서 균일화된 임의의 작업은 아님
* 적절한 척도를 사용해 하이퍼파라미터 탐색해야

범위 지정(ex. 은닉노드 수, 층 개수) / 범위 지정 + 로그 스케일(더욱 균등하게) (ex. 학습률)  
/ 범위 지정 + 로그 스케일 + 지수 가중 평균(ex. 베타) -> B값은 1에 가까울수록 민감

[배치 정규화] – 입력 기능을 정규화하면 학습 속도 높임

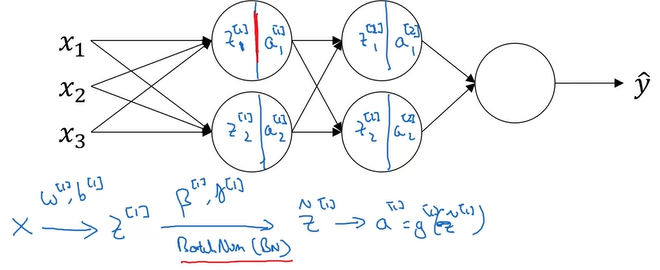
a[2]은 다음 층(w[3], b[3])의 입력이기에 훈련에 영향을 준다. -> a[2]가 아닌 z[2]를 정규화

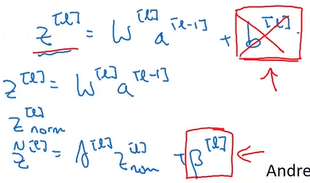
배치 노름 도입 (층[l] 생략) : 평균값, 분산 계산하고, 값을 빼고 나누는 것

  
  
  
  
  
  
  
모델의 학습 가능한 파라미터(r, B) => RMS prop, Adam을 이용

IF r(gamma) = sqrt(시그마^2 + epsilon), B(Beta) = 뮤, -> z틸다(i) = z(i)

입력층 뿐만 아니라 신경망의 일부 은닉층까지 정규화를 적용하는데, 은닉 단위가 강제적으로 평균0, 분산1이 되는 것을 원치 않는다. (비선형 활성화함수 원해서)  
=> r, B 조정해서 은닉 단위 값(z(i))들의 일부 고정된 평균과 분산을 갖도록 정규화 (조정 안하면 은닉 유닛은 항상 평균0, 분산1)

  
첫번째 은닉층에 맞는 입력 X   
🡪 z[1] 값 산출   
🡪 배치노름 적용해   
 정규화된 틸다 z[1]   
🡪 활성화함수 a[1] = g[1](틸다z[1])  
🡪 z[2] 값 산출

배치 노름은 훈련셋의 미니 배치와 같이 적용된다. (배치노름은 한 번에 하나의 미니배치로 처리)  
  
- z[l] 값들에 대해서 배치 노름은 평균값을 0으로 만들기 때문에 b[l] 파라미터를 갖는 의미 없다.  
-> B(Beta)[l]로 대체 : 이동이나 바이어스 항들에 영향을 주는 파라미터

z[l]의 차원 : (n[l], 1) b[l]의 차원 : (n[l], 1)   
B[l], r[l]의 차원 : (n[l], 1) – 각각 은닉에 대해 B[l], r[l]을 통해 평균값과 분산이 스케일링되기에 n[l]

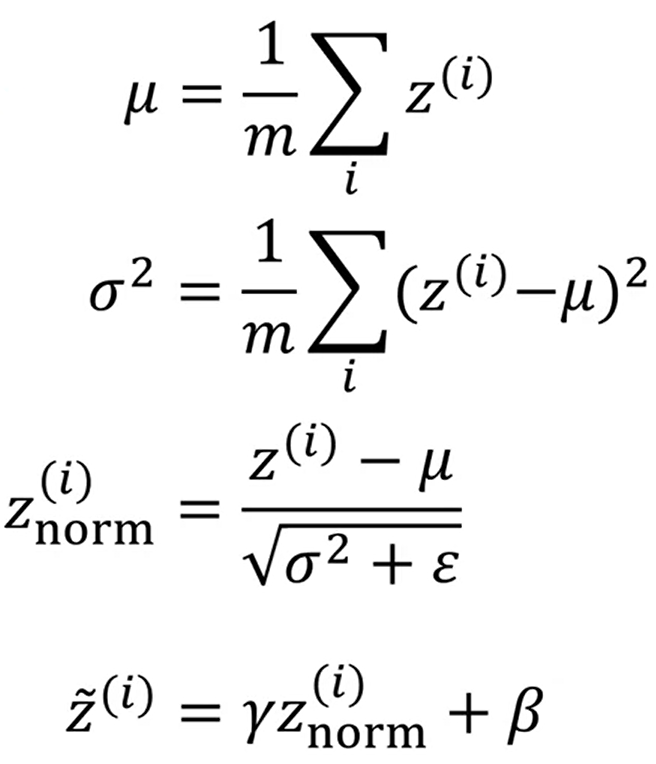
배치 노름이 잘 작동하는 이유  
- 입력특성을 정규화하여 학습의 속도 증가시킴 (입력뿐만 아니라 은닉층에도 정규화 적용)  
- 정규화를 통해 ,W 변화에 덜 민감해짐

데이터의 분포가 변함 : 공변량 변화 (ex. 검은 고양이 사진 로지스틱회귀로 분류하는 것을 유색 고양이 에 적용하면 분류기가 잘 동작하지 않을 수 있어 다시 학습시켜야 할 수 있다.)

파라미터가 변하면 은닉 단위의 값은 항상 변하는 것이고, 공변량 변화 문제가 생김  
-> 배치 노름을 적용해 평균/분산이 같게 해주어 은닉 단위 값의 분포가 이동하는 양을 줄여준다.

일반화 효과  
- 미니배치로 평균/분산 계산했기에 전체 데이터로 구했을 때보다 더 잡음이 많다.  
- 드롭아웃도 은닉층 활성화에 잡음을 더한다.  
=> 더 큰 미니배치 크기 이용하면, 잡음을 줄이고 일반화 효과를 줄인다.

[테스트셋에서 배치노름]

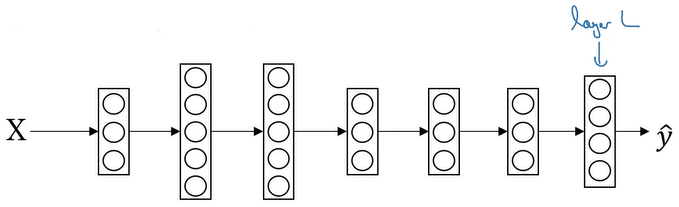
m : 미니배치 개수  
test에선 미니배치가 없을 수 있다.(한번에 처리)  
-> 뮤와 시그마제곱을 계산하는 다른방법 필요  
\* 일반적인 배치 정규화는 지수 가중 평균을 이용해 미니 배치에서의 값으로 예측하여 구함

최종 네트워크를 통해 전체 미니배치에서 전체 훈련셋을 실행시켜 뮤와 시그마제곱을 가지지만, 지수 가중 평균을 도입해 뮤와 시그마제곱값을 추적한다. <running average>

[소프트맥스] – C 중 하나 또는 여러 클래스 중 하나를 인식해 예측  
(입력을 범주화하려는 클래스의 수 = C)

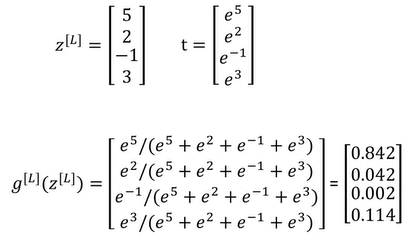
다른 결과값들에 거쳐 정규화가 되어야 한다. => 벡터의 값을 갖고, 결과값도 벡터로 가져야 함

출력층(output layer)의 노드 수 n[l] = C  
ex) C = 4 // 첫번째 노드는 C=0일 확률, … , 마지막 네번째 노드는 C=3일 확률 -> 확률의 합=1  
결과값 y\_hat : (4, 1) 차원의 벡터

마지막 층 L  
z[L] = w[L] \* a[L-1] + b[L]  
-> z[L]에 softmax 적용

t = e^(z[L]) (shape : (4, 1))  
a[L] = e^(z[L]) / sum(t\_i) = t\_i / sum(t\_i) = y\_hat = g[L](z[L]) (shape : (4, 1))

* 시그모이드 : 실수를 입력하고 실수를 출력
* 소프트맥스 : 다른 결과값들에 거쳐 정규화가 되고, 벡터를 입력하고 벡터로 출력

= a[L]

* hard max : 벡터 z를 가져와서 이 벡터와 일치시킴   
  => z의 요소들 중 가장 큰 요소에 1을 넣고 나머진 0으로 채움

소프트맥스가 로지스틱 활성화 함수를 두개의 클래스가 아닌 C 클래스로 일반화 함  
IF C = 2, 로지스틱 회귀분석으로 됨 -> 단일 출력을 계산하는 방식으로 축소됨

ex) y = [0 1 0 0] (ground truth)  
y\_hat = [0.3 0.2 0.1 0.4] => 신경망이 잘 예측하지 못한 것이다.

손실함수 = -(y1 \* log(y1\_hat) + y2 \* log(y2\_hat) + y3 \* log(y3\_hat) + y4 \* log(y4\_hat))  
 = - y2 \* log(y2\_hat) = - log(y2\_hat)  
\*\* y1 = y3= y4 = 0

손실함수를 최소화하려면 y2\_hat을 크게 해야 하는데, 가장 큰 값이 아니다.

역전파 dz[L] = y\_hat – y

[tensorflow 코드]

