# 12. Training Neural Network
# 2019년 가을 학기

2019년 08월 18일
https://www.github.com/KU-BIG/KUBIG_2019_Autumn

*이 글은 사이토 고키의 '**밑바닥부터 시작하는 딥러닝 : 파이썬으로 익히는 딥러닝 이론과 구현**과 박희원님의 슬라이드[1]를 많이 참고하였다.*

이번 강의에서는 기본적인 딥러닝 모델들을 적용할 때 어떻게 하면 효과적으로 학습할 수 있을지에 대해서 설명한다. 요즘에는 딥러닝 프레임워크들이 워낙 잘 구현되어있어 원리를 잘 몰라도 코드를 돌릴 수 있지만 학습을 잘하기 위해서 어떤 방법을 기본적으로 사용하는지 이해한다면 모델을 튜닝하는데 도움이 될 수 있다고 생각한다. 미니배치, 가중치 매개변수 초기값, 배치 정규화, Overfitting 방지법(가중치감소, 드랍아웃)을 설명하겠다.

## 1. Introduction

딥러닝과 MLP(Multi Layer Perceptron)의 가장 큰 차이점은 layer의 깊이다. MLP의 층이 깊어진 것이 딥러닝이기 때문이다. 즉, 딥러닝의 핵심은 hidden layer의 결합에 있다. 물론, 대다수의 모델과 같이 딥러닝에서도 초기값의 설정은 매우 중요하다. 좋은 초기값을 고를수록 연산량이 줄어들기 때문이다.

이 때문에 딥러닝을 할 때 어떻게 parameter를 설정할 지가 고민이다. 몇 개의 hidden layer를 설정할 것이고 각 hidden layer마다 node(neuron)를 몇 개로 둘 것인지 고민할 것이다. 더불어 어떤 활성화 함수(activation function)를 쓸 것인지도 주어져 있지 않다. 하지만 고민없이 이러한 부분들을 선택하게 되면 후에 고생하게 된다.

대표적으로 hidden layer의 개수에 대해서 다루겠다. 단순하게 생각하면, hidden layer가 많아질수록 복잡한 형태의 decision boundary가 형성될테니 좋은 성능을 보일 것이다. 하지만 이렇게 되면 전 강의에서 지적한 대로 세상의 쓴 맛을 맛보게 될 것이다. Overfitting(과적합), time complexity(느리다), vanishing gradient(덜하다)가 당신을 두 팔 벌려 환영할 것이다.

[1] https://www.slideshare.net/HeeWonPark11/ss-80653977

이 세 문제에 대한 해결방안은 다음과 같다 :

## 1.1 Overfitting – too many neurons. 뉴런이 너무 많을 때.

다다익선이라는 말은 적어도 neural network에는 통하지 않는다. 사실 이러한 overfitting 문제는 데이터 분석 모델 전반에 나타나는 문제이다. 뉴런이 너무 많으면 왜 좋지 않은 network라고 하는가. 결론적으로 해당 network를 통해서는 train data(학습 데이터)에**'만'** 잘맞는 모델이 생성된다.

Train data에만 잘 맞는 모델이 왜 문제인가라는 의문이 들 수 있다. 앞선 강의들에서 강조했듯이 우리는 train data를 예측하고 싶은게 아니기 때문에 overfitting은 문제가 된다. 이미 답을 아는 data를 예측해서 무엇하겠는가. 우리는 train data가 아닌 다른 새로운 데이터, test data를 예측하는 것이 목표이다. 이러한 상황에서는 train data에 너무 잘 맞는 분석의 나머지 다른 test data의 예측 정확성이 떨어지는 모델은 쓸모가 없다.

그러면 overfitting 대한 해결 방안은 무엇일까? 바로 '**Drop out**'이라는 방법이다. 많은 neuron들이 들어있는 복잡한 network를 더 단순하게 만들기 위해서 random하게 neuron들을 drop out, 즉 **버린다**. 물론 모든 neuron을 학습하는 것보다 train set에 대한 설명력이 떨어지지만 성능은 말이 안되게 개선된다. 버릴 neuron과 살려둘 neuron은 Bernoulli에 의해 결정되기 때문에 뉴런의 개수가 n개라면, 살아남는 neuron은 독립적인 Binomial에 의해 결정된다.


https://www.slideshare.net/HeeWonPark11/ss-80653977

## 1.2 Too slow - 계산이 너무 오래 걸린다.

우리의 목적은 bias와 weight의 최적값을 찾는 것이다. 현재 가중치로 산출된 cost function의 gradient를 구해서 cost function의 값을 낮추는 방향으로 update하는 누구나 알 법한 그 방법,

**gradient descent**를 이용한다. 만인이 사랑하는 gradient descent이지만 딥러닝이 요구하는 계산량을 수행하기에는 너무 시간이 오래 걸린다는 문제가 있다.

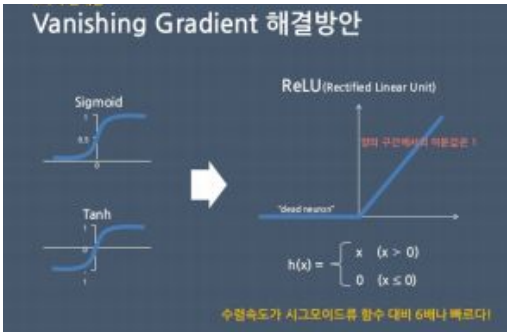
딥러닝에선 다른 optimizer로 이를 어느정도 해결한다. 대표적인 optimizer로는 SGD(Stochastic Gradient descent) 확률적경사하강법이 있다. 이에 대한 자세한 수식과 내용은 담지 않지만 전반적으로 파악을 하자면 보통의 경사하강법과는 다르게 이름 그대로 확률적으로 무작위로 골라낸 데이터에 대해 경사하강법을 수행하는 것이다. SGD의 두드러지는 단점으로는 랜덤 확률이다 보니 값이 불안정하게 흔들리는 경우가 있고 수렴 안정성이 낮아 최적해를 찾을 수 없을 확률이 높다. 이에 대한 대안으로 학습 속도와 운동량을 고려한다.

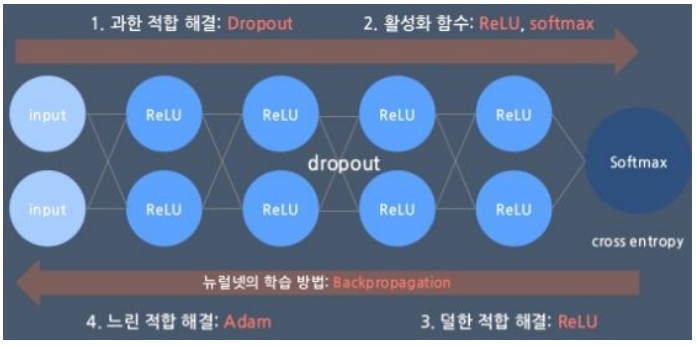
속도 중심의 다른 optimizer는 adagrad(가본 곳은 더 정밀하게, 안 가본 곳은 대충), adadelta (계속 정밀. 모든 것을 탐색하지 않고 끊음), RMSprop(adagrad에서 발전함. 가본 곳은 더 정밀하게 가지만 그 순간에 gradient를 재계산하여 속도 조절) 등이 있다.

운동량에 중심을 둔 optimizer는 momentum(물리에서 따옴. 절대 아닐 것 같은 길이 아닌 진짜 같은 곳으로), NAG(momentum에서 나아가 순간 순간 gradient를 재계산하여 운동량 높임) 등이 있는데 현재로써 가장 보편적이고 성능이 우수한 optimizer는 adam이지만 다양한 알고리즘을 이용하여 많은 모델을 적합하는 것이 중요하다.

## 1.3 Vanishing gradient - gradient가 사라진다.

Relu activation function을 이용하여 해결한다. Relu는 입력값이 음수일 때 0을 출력하고, 그렇지 않으면 자기자신을 출력하는 함수이다. 미분계수가 0과 1의 값만 가지게 되므로, 학습속도 정체의 문제가 발생하지 않는다는 장점을 가진다. 그러나 음수에서 미분계수가 0이기 때문에, 음수의 입력값이 주어질 때 학습이 이루어지지 않는다는 단점이 있다.




https://www.slideshare.net/HeeWonPark11/ss-80653977

# 2. 학습을 위한 기술

## 2.1 미니배치 학습

기계학습 문제는 train data를 사용해 학습한다. 더 구체적으로는 train data에 대한 손실함수의 값을 구하고, 그 값을 최대한 줄여주는 매개변수를 찾아낸다. 이렇게 하려면 모든 train data를 대상으로 손실함수 값을 구해야한다. 즉, train data가 100개 있으면 그로부터 계산한 100개의 손실함수 값들의 합을 지표로 삼는다.

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk}\log y_{nk}$$

데이터가 N개라면 $t_{nk}$는 n번째 데이터의 k번째의 값을 의미한다. ($y_{nk}$는 신경망의 출력, $t_{nk}$는 정답 레이블이다.) 위의 수식은 데이터 하나에 대한 손실함수를 단순히 N개로 확장한 후 N으로 나누어 정규화하고 있다. N으로 나눔으로써 '평균 손실함수'를 구하는 것이다. 이렇게 평균을 구해 사용하면 train data 개수와 관계없이 언제든 통일된 지표를 얻을 수 있다.

그런데 딥러닝이 학습하는 빅데이터의 수준은 수백만에서 수천만이 넘는 거대한 값이 되기도 하다. 이 많은 대상을 일일히 손실함수를 계산하여 합하는 것은 현실적이지 않다. 이런 경우 데이터 일부를 추려 전체의 '근사치'로 이용할 수 있다. 이 일부를 미니배치라고 한다. 가령 60,000장의 train data 중에서 100장을 무작위로 뽑아 그 100장을 사용하여 학습하는 것이다. 이런 학습 방법을 미니배치 학습이라고 한다.

## 2.2 가중치의 초기값

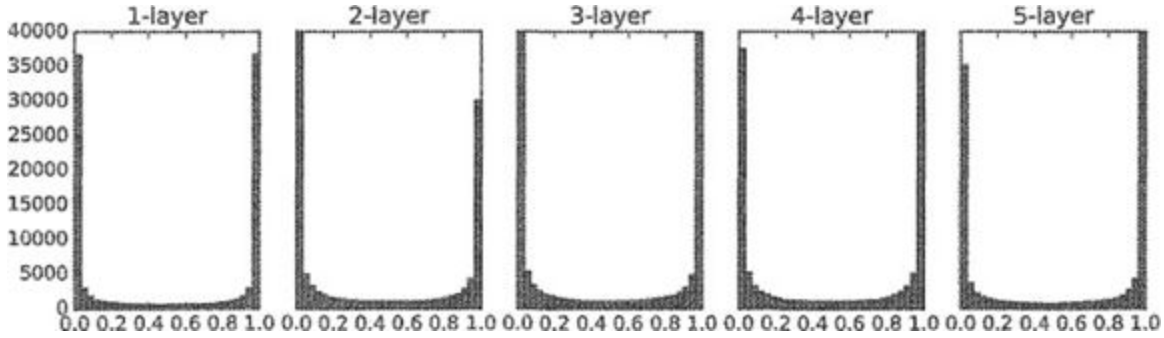
신경망 학습에서 특히 중요한 것이 가중치의 초기값이다. 가중치의 초기값을 무엇으로 설정하느냐가 신경망 학습의 성패를 가르는 일이 실제로 자주있다. 권장하는 초기값에 대해서 설명하고 실제로 신경망 학습이 신속하게 이루어지는 모습을 확인하자.

### 2.2.1 초기값을 동일하게 하면

가중치 W를 모두 같은 값으로 초기화 한다면 어떻게 될까? 순전파(Forward Propagation)때는 두번째 층의 뉴런에 모두 같은 값이 전달되고, 역전파(Back Propagation)때 미분의 연쇄법칙에 의해 두번째 층의 가중치가 모두 똑같이 갱신된다. 다시 말해 뉴런의 개수와 상관없이 같은 값을 출력하게 돼 네트워크의 표현력을 제한하게 된다.

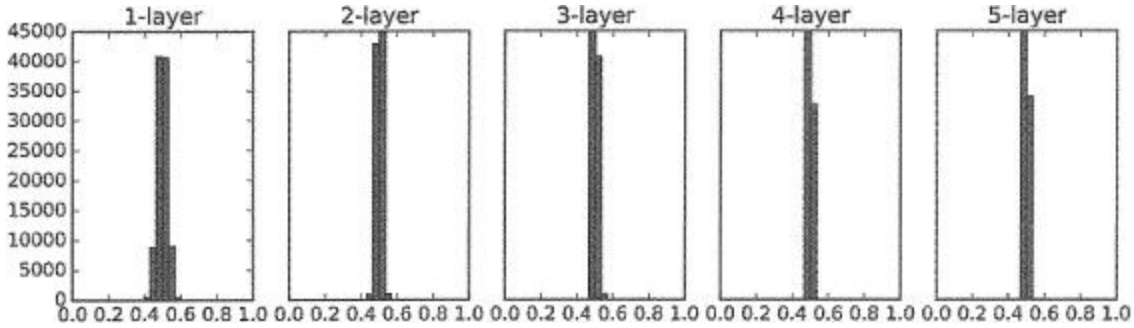
### 2.2.2 은닉층의 활성화값 분포와 문제점

가중치의 초기값에 따라 은닉층의 활성화값들이 어떻게 변하는지 알아보자. 신경망에 무작위로 생성한 데이터를 흘리며 각 층의 활성화값 분포를 히스토그램으로 그려보겠다. 5개의 층에 각각 100개의 뉴런이 있으며 입력데이터로 1000개의 정규분포 난수를 생성하여 이 신경망에 흘린다. 활성화 함수로는 시그모이드 함수를, 가중치의 분포는 표준편차가 1인 정규분포를 사용했다. 이때 활성화값들의 분포가 어떻게 되는지 관찰해보자.




각 층의 활성화값들이 0과 1에 치우쳐 분포됨을 알 수 있다. 여기에서 사용한 시그모이드 함수는 그 출력이 0이나 1에 가까워지면 미분이 0에 다가가기 때문에 데이터가 0과 1에 치우치게 되면 역전파의 기울기 값이 점점 작아지다 사라지게 된다. 이는 기울기 소실(Vanishing Gradient)이라 알려진 문제이다. 층을 깊게 하는 딥러닝에서 기울기 소실은 심각한 문제가 될 수도 있다.

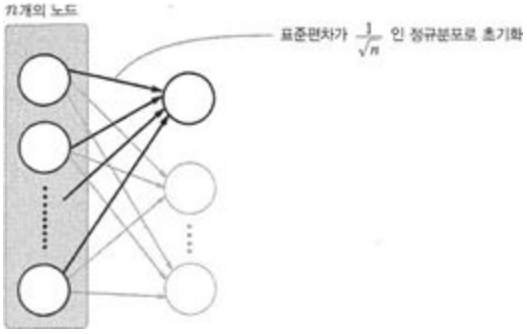
표준편차를 0.01로 한 정규분포의 경우 각 층의 활성화값 분포는 다음과 같이 된다.

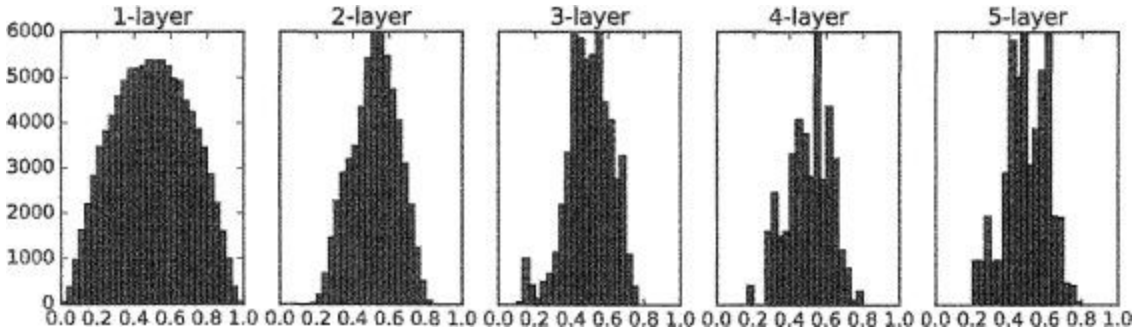



이번에는 0.5 부근에 집중되었다. 앞의 예처럼 0과 1로 치우치진 않았으니 기울기 소실 문제는 일어나지 않다만, 활성화값들이 치우쳤다는 것은 표현력 관점에서는 큰 문제가 있다. 이 상황에서는 다수의 뉴런이 거의 같은 값을 출력하고 있으니 뉴런을 여러 개 둔 의미가 없어진다. 예를 들어 뉴런 100개가 거의 같은 값을 출력한다면 뉴런 1개짜리와 별반 다를게 없다. 그래서 활성화값들이 치우치면 표현력을 제한다는 관점에서 문제가 된다.

### 2.2.3 Xavier 초기값

이러한 기울기 소실 문제와 표현력의 제한 문제를 덜 일으킨다고 알려진 초기값을 알아보자. 자비에르(Xavier)가 논문에서 권장한 Xavier 초기값이다. Xavier 초기값은 일반적인 딥러닝에서 표준적으로 이용하고 있다. 자비에르는 각 층의 활성화 값들을 광범위하게 분포시킬 목적으로 가중치의 적절한 분포를 찾고자 했다. 그리고 앞 계층의 노드가 n개라면 표준편차가 $\frac{1}{\sqrt{n}}$ 인 분포로 사용하면 된다는 결론을 이끌었다.

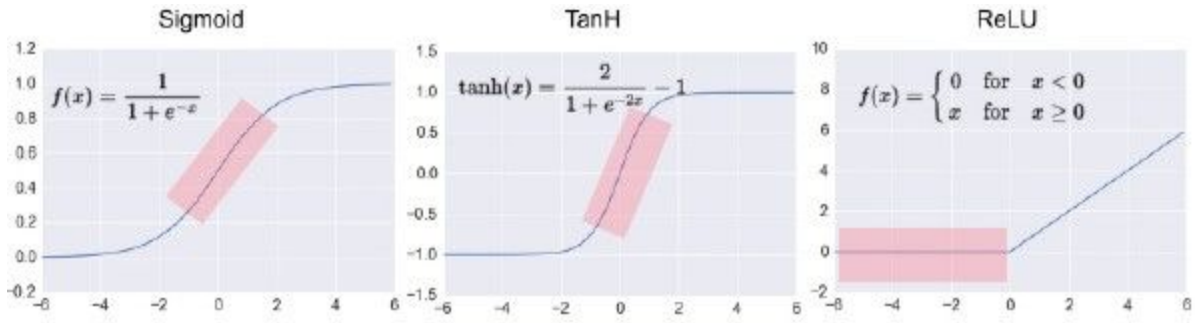


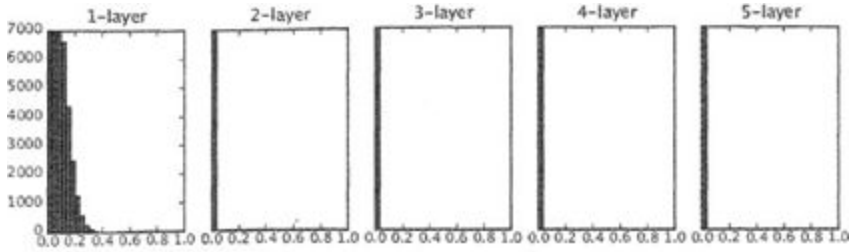

Xavier 초기값을 사용한 결과는 위와 같다. 이 결과를 보면 층이 깊어질수록 형태가 다소 일그러지지만, 앞에서 본 방식보다는 확실히 넓게 분포됨을 알 수 있다. 각 층에 흐르는 데이터는 적당히 퍼져 있으므로, 시그모이드 함수의 표현력도 제한받지 않고 학습이 효율적으로 이뤄질 것으로 기대된다.

### 2.2.4 He 초기값

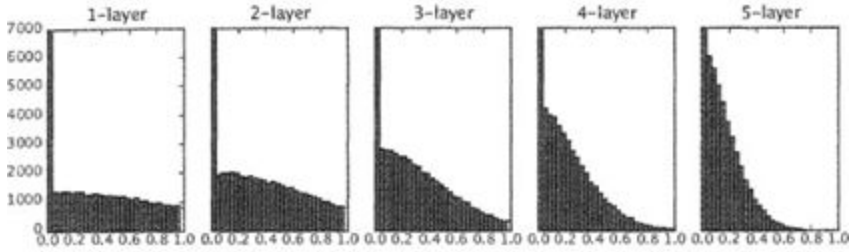
Xavier 초기값은 sigmoid 함수와 tanh함수에 알맞게 설정된 가중치 초기값이다. 반면 ReLU를 이용할 때는 ReLU에 특화된 초기값을 이용하라고 권장한다. 이 특화된 초기값을 찾아낸 사람의 이름을 따 He(히) 초기값이라 한다.



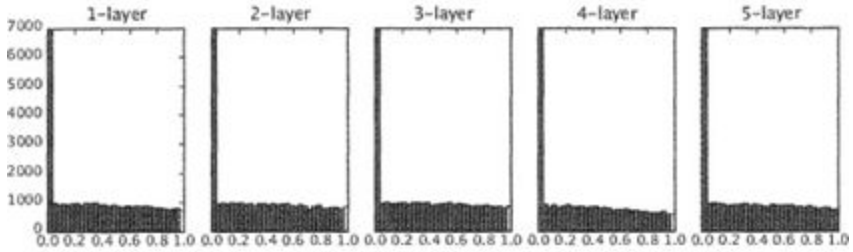

He 초기값은 ReLU 함수에 맞게 Xavier 초기값을 변형했다. Xavier 초기값은 활성화 함수가 선형인 것을 전제로 하는데 Sigmoid와 tanh는 중앙 부근이 거의 선형인 함수이므로 Xavier 초기값이 적당하다. 반면 ReLU는 음의 영역 값이 0이라서 더 넓게 분포시키기 위해 계수가 클 필요가 있다. 따라서 Xavier 초기값의 표준편차에 2를 곱하여 더 넓은 정규 분포를 만들어 주었다.




표준편차가 0.01인 정규분포를 가중치 초깃값으로 사용한 경우




Xavier 초깃값을 사용한 경우




He 초깃값을 사용한 경우

결과를 보면 std = 0.01일 때의 각 층의 활성화값들은 아주 작은 값들로 이루어졌다. 신경망에 아주 작은 값이 흐른다는 것은 역전파 때 가중치의 기울기 역시 작아진다는 뜻이다. 이는 중대한 문제이며 실제로도 학습이 거의 이뤄지지 않을 것이다.

이어서 Xavier 초기값 결과를 보면 층이 깊어지면서 치우침이 조금씩 커진다. 실제로 층이 깊어지면 활성화값들의 치우침도 커지고, 학습할 때 '기울기 소실' 문제를 일으킨다. 마지막으로 He 초기값은 모든 층에서 균일하게 분포된다. 층이 깊어져도 분포가 균일하게 유지되기에 역전파 때도 적절한 값이 나올 것으로 기대할 수 있다. 즉, 활성화 함수로 ReLu를 사용할 때는 He 초기값을, sigmoid나 tanh 등의 S자 모양의 곡선일 때는 Xavier 초기값을 쓰면 좋다.

## 2.3 배치 정규화 (Batch norm)

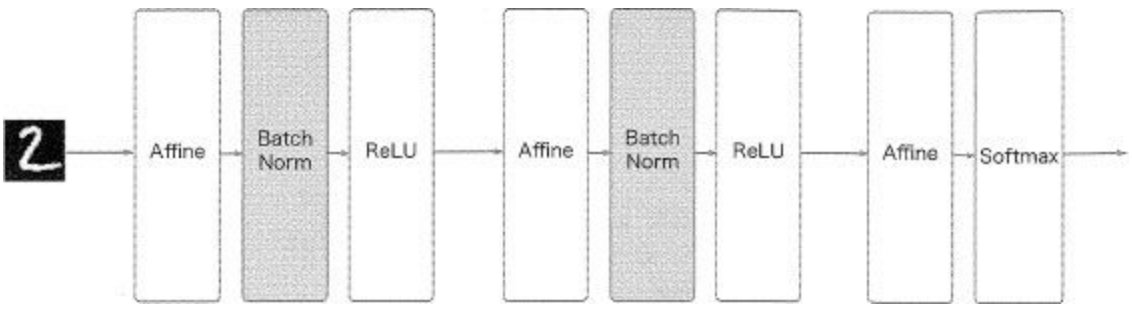
앞 장에서 가중치의 초기값을 적절히 설정하면 각 층의 활성화값 분포가 적당히 퍼지면서 학습이 원활하게 수행되는 것을 보았다. 그렇다면 각 층이 활성화값을 강제로 퍼뜨리면 어떨까? "배치 정규화"가 그런 아이디어에서 출발한 방법이다.

배치 정규화가 주목받는 이유는 다음과 같다.

- 학습을 빨리 진행할 수 있다.
- 초기값에 크게 의존하지 않는다
- overfitting을 억제한다

딥러닝의 학습시간이 길다는 걸 생각해보면 첫번째 이점은 아주 반가운 일이다. 초기값에 크게 신경 쓸 필요가 없고, overfitting 억제 효과가 있다는 점도 딥러닝 학습의 두통거리를 덜어준다.

배치 정규화의 기본 아이디어는 앞에서 말했듯이 각 층에서의 활성화값이 적당히 분포되도록 조정하는 것이다. 그래서 다음 그림과 같이 데이터 분포를 정규화하는 '배치 정규화 계층'을 신경망에 삽입한다.



배치 정규화는 그 이름과 같이 학습 시 미니배치를 단위로 정규화한다. 구체적으로는 데이터 분포가 평균이 0, 분산이 1이 되도록 정규화한다.

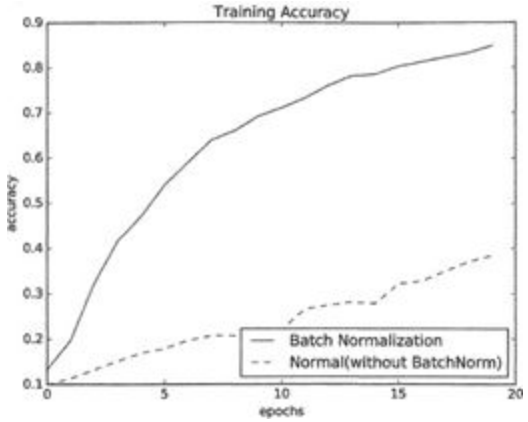
$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

미니배치 B = $\{x_1, x_2, ..., x_m\}$라는 m개의 입력 데이터의 집합에 대해 평균 $u_B$와 분산 $\sigma_b^2$을 구한다. 그리고 입력 데이터를 평균이 0, 분산이 1이 되게 정규화한다. 단순히 미니배치 입력 데이터 $\{x_1, x_2, ..., x_m\}$을 평균 0, 분산 1인 데이터 x-hat으로 변환하는 일을 활성화 함수의 앞에 삽입함으로써 데이터 분포가 덜 치우치게 할 수 있다.

### 2.3.1 배치 정규화의 효과




MNIST 데이터셋을 사용하여 배치정규화 계층을 사용할 때와 사용하지 않을 때의 학습 진도가 어떻게 달라지는지를 나타낸 그래프이다. 그래프와 같이 배치 정규화가 학습을 빨리 진전시킨다.

## 2.4 Overfitting 해결책

Overfitting이란 신경망이 train data에만 지나치게 적합하여 그 외 데이터에는 제대로 대응하지 못하는 상태를 말한다. train data에는 포함되지 않은, 아직 보지 못한 데이터가 주어져도 바르게 식별해내는 모델이 바람직하다. 딥러닝 기법을 이용할 시 복잡하고 표현력이 높은 모델을 만들 수 있는 만큼 overfitting을 억제하는 기술이 중요해진다.

### 2.4.1 가중치 감소

Overfitting 억제용으로 예로부터 많이 이용해 온 방법 중 가중치 감소(weight decay)가 있다. 이는 학습 과정에서 큰 가중치에 그에 상응하는 큰 페널티를 부과하여 overfitting을 억제하는 방법이다. 원래 overfitting은 가중치 매개변수의 값이 커서 발생하는 경우가 많기 때문이다.

가장 일반적인 regularization기법인 L2 Regularization을 소개하겠다. 기존 손실함수($L_{old}$) 에 모든 학습 파라미터의 제곱을 더한 식($\|\mathbf{W}\|_2^2$)을 새로운 손실함수($L_{new}$)로 쓴다. 여기에서 1/2이 붙은 것은 미분 편의성을 고려한 것이고, λ는 패널티의 세기를 결정하는 사용자 지정 hyper-parameter이다. 이 기법은 큰 값이 많이 존재하는 가중치에 제약을 주고 가중치값을 가능한 널리 퍼지도록하는 효과를 내어 overfitting을 억제한다.

$$W = [w_1 \quad w_2 \quad \dots \quad w_n]$$

$$L_{new} = L_{old} + \frac{\lambda}{2}(w_1^2 + w_2^2 + \dots + w_n^2)$$