딥러닝 학습 시 **Overfitting** 문제 해결을 위한 **Batch Normalization** 구현 검증

3학년 2 반 16번 이름 : 정 휘 준

Ⅰ. 연구의 필요성 및 목적

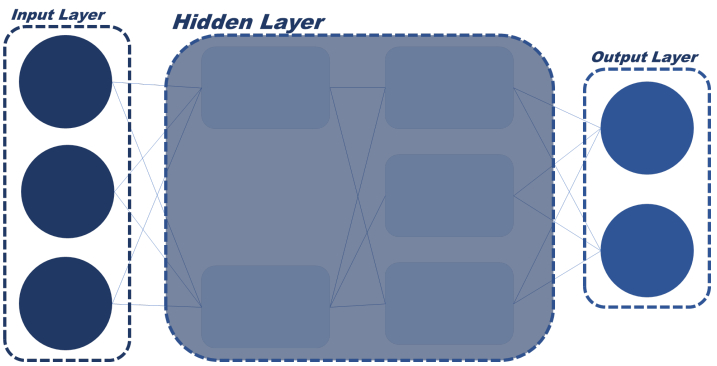
Overfitting은 딥러닝 학습단계에서 training data set에 대해 속속들이 알지 못하는 경우가 대부분이기 때문에 해결이 힘들고 인공지능 개발자들이 특히 골머리를 앓는 문제이다. 본 연구에서는 이 overfitting 문제 해결을 위한 방법 중 하나인 batch normalization을 구현으로 검증해 보고자 한다.

Ⅱ. 이론적 배경

1. Deep Learning Basic

1.1. Neural Networks

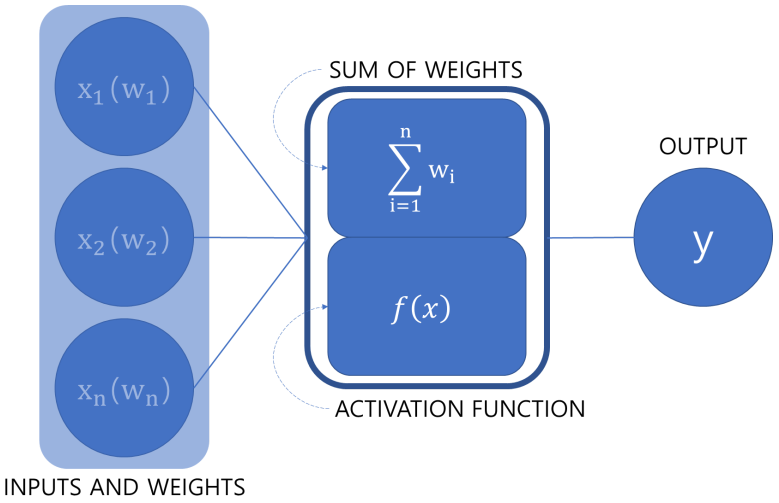
Neural Networks란 AI분야의 주요 문제를 해결하기 위해 컴퓨터 과학과 통계를 접목하면서 생물학적 뉴런이 서로 간에 신호를 보내는 방식을 모방한다. Neural Networks는 하나의 입력 레이어, 하나의 출력 레이어 그리고 하나 이상의 히든 레이어가 노드 방식으로 연결되어 있다. 이렇게 구성된 neural networks는 현실 세계의 복잡한 문제를 해결할 수 있다. Neural Networks는 훈련 데이터에 의존하여 학습하고 시간이 지나면서 자체적으로 정확도를 업데이트한다.



[그림 1] Neural Network

1.2. Perceptron

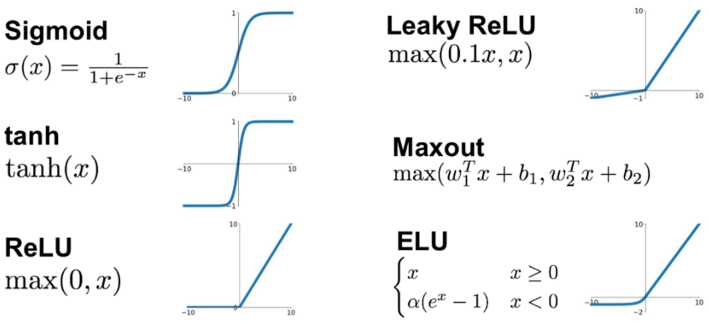
Perceptron은 무엇인가를 인지하는 능력 [Perception]과 신호를 받아 들여 정보를 전달하고 처리하는 [Neuron]의 합성어이다. 즉 perceptron이란 생물학적 뉴런이 감각정보를 받아서 문제 해결하는 방식을 모방한 인공 뉴런이다. perceptron은 입력값(input)과 가중치(weight)를 받아 activation function으로 계산한 후 출력값을 내보낸다. 가중치가 큰 입력값은 출력값을 결정하는 데에 역할이 크다는 뜻이 된다. 또한 perceptron의 출력값은 이진수이므로 0 또는 1이다. 그래서 perceptron이 해결할 수 있는 가장 간단한 문제는 분류이며 activation function의 조정하여 perceptron의 출력값이 넓은 범위의 숫자로 출력되도록 변경하면 종속변수와 독립변수 간의 관계를 이해하기 위한 회귀 문제도 해결이 가능하다.



[그림 2] Perceptron의 구성 요소

1.3. Activation Function

activation function은 perceptron의 출력값이며 neural network를 개발하는 사람이 신경망의 예측 성능이 좋게 만드는 함수를 선택한다. 마지막 단계에 있는 activation function은 전체 neural network의 출력값을 결정하기 때문에 매우 중요한 의미를 갖는다. 딥러닝으로 해결하려는 문제는 사실 대부분 단순한 문제들이 아니기 때문에 데이터들은 다차원 공간에서 굉장히 비선형적인 모습을 보여줄 것이다. 여기에 activation function은 단순히 0과 1로 나뉘는 세계보다 더 비선형적 특징을 처리할 수 있게 하는 역할을 하는 함수이다.



[그림 3] Activation Function의 종류

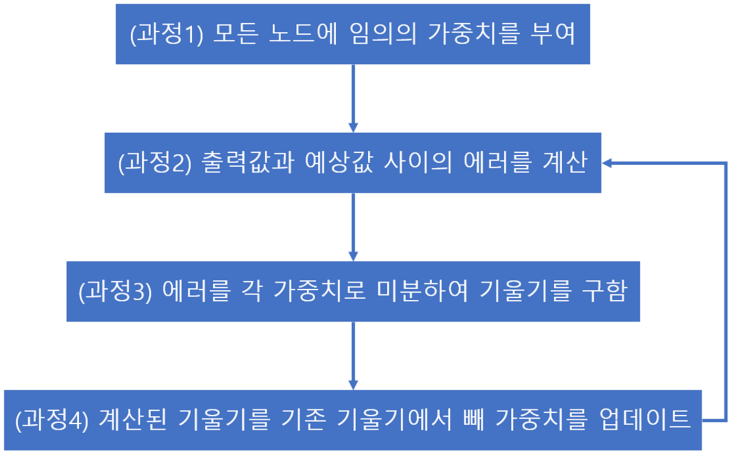
1.4. Supervised Learning

Machine Learning의 학습 방법은 크게 supervised learning과 non-supervised learning으로 나뉜다. Supervised Learning이 일반적이며 구현하기도 쉽다고 한다.

Supervised Learning은 데이터를 분류하거나 결과를 정확하게 예측하는 알고리즘을 훈련하기 위해 레이블이 지정된 data set들을 사용한다. 학습결과의 성과가 좋으려면 훈련 데이터의 양이 많아야 하고 훈련 데이터가 범용성(generalization)을 갖고 있어야 한다. Supervised Learning은 이미지 개체 인식, 예측 분석, 고객 감정 분석, 스팸 감지 등의 분야에 활용되고 있지만 특정 분야의 supervised learning model을 정확하게 구성하려면 특정 수준의 전문지식이 필요할 수 있고 data set에 유효하지 않은 정보가 많으면 알고리즘이 잘못 학습될 가능성이 높으며 supervised learning model을 교육하는데 많은 시간이 소요될 수도 있다. 또한 non-supervised learning에서는 가능하지만 supervised learning model에서는 데이터를 자체적으로 클러스터링하거나 분류할 수가 없다.

2. Gradient Vanishing

2.1. Supervised Learning 알고리즘



[그림 4] Supervised Learning의 알고리즘

Neural Network에서는 이전 perceptron 출력이 다음 perceptron의 입력값이 되어 가중치의 합과 activation function의 결과값이 네트워크를 따라 계속 이루어지며 [그림 4]의 (과정1) 에서는 마지막 perceptron의 activation function의 결과값이 예측치가 되는데 이 단계에서는 무작위로 가중치를 부여하므로 예측치는 절대로 맞는 값이 아니므로 (과정2)에서는 예측치와 activation function의 결과값의 에러를 계산하여 (과정3)에서는 에러를 최소화로 관리하기 위하여 Loss Function으로 에러를 관리하게 되며 가중치를 조금씩 변경한다. 그 원리는 이렇다. 각 가중치들이 적용된 현재 지점에서의 기울기는 기울기의 반대방향으로 향하는 것이 에러의 값을 감소시키는 방법임을 알 수 있다. 즉 현재의 기울기의 반대방향으로 가중치를 이동하면 무조건 에러는 감소하게 된다. 그렇다면 가중치는 얼마만큼 반대방향으로 이동해야할까 하는 문제가 있는데 많은 값을 이동하면 오히려 에러가 더 커질 수 있으니 작은 값을 선택하여 기울기의 반대방향을 곱한만큼

가중치를 업데이트하는 (과정4)를 거친 다음 다시 (과정2)부터 시작하여 에러값이 최소가 될 때까지 반복하는 과정을 거친다.

2.2. Gradient Vanishing

2.2.1. Gradient Vanishing의 의미

딥러닝 분야에서 레이어를 많이 쌓으면 오히려 학습이 잘 되지 않을 수가 있는데 [그림 3]의 (과정4)에서 (과정2)로 되돌아가 가중치를 업데이트하는 과정을 back-propagation이라고 하고 출력과 멀어질수록 gradient가 매우 작아지는 현상을 gradient vanishing이라고 한다.

2.2.2. Gradient Vanishing의 원인

2.2.2.1. $sigmoid$ 함수

[그림 5] sigmoid함수와 sigmoid의 도함수

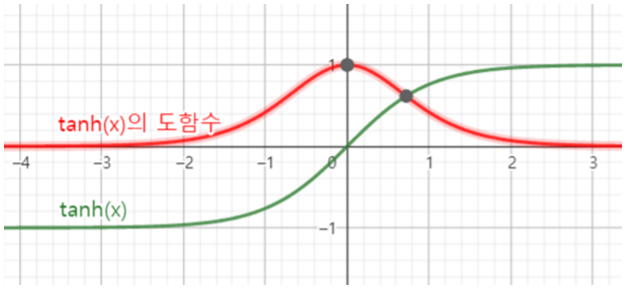
$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d}{dx}sigmoid(x) = \frac{d}{dx}(1+e^{-x})^{-1} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = sigmoid(x)(1 - sigmoid(x))$$

$sigmoid$ 함수는 모든 실수 입력값을 0보다 크고 1보다 작은 미분 가능한 수로 변환하는 특징을 가지며 $sigmoid$ 함수의 결과값은 확률 형태이기 때문에 결과를 확률로 해석할 때 적용하면 유리하다. $sigmoid$ 함수는 음수값을 0에 가깝게 표현한다. 또한 $sigmoid$의 도함수의 최댓값은 0.25인데 back-propagation에서 $sigmoid$ 함수의 미분값이 거듭 곱해지면 출력층과 멀어질수록 gradient vanishing이 생길 수 밖에 없다.

2.2.2.2. $tanh$ 함수



[그림 6] $tanh$함수와 $tanh$의 도함수

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

$$\frac{d}{dx}tanh(x) = \frac{d}{dx}\left(\frac{e^x-e^{-x}}{e^x+e^{-x}}\right) = \frac{\left(e^x+e^{-x}\right)^2-\left(e^x-e^{-x}\right)^2}{\left(e^x+e^{-x}\right)^2}$$

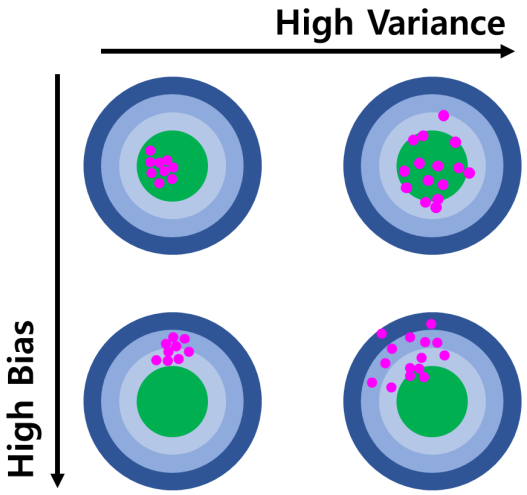
$$= 1 - \left(\frac{e^x-e^{-x}}{e^x-e^{-x}}\right)^2 = 1 - tanh^2(x)$$

back-propagation 과정에서 $sigmoid$ 함수의 층이 많아질수록 학습이 효과적이지 않은 한계점을 개선하기 위해 $tanh$를 사용하게 되었는데 $tanh$함수는 출력값이 [-1,+1]의 범위를 갖도록 출력값의 범위가 2배 늘었고 $tanh$의의 도함수의 최댓값은 1이다. $sigmoid$의 도함수와 비교하면 최댓값이 4배가 차이나는 것을 알 수 있다. 그럼에도 x값이 크거나 작아짐에 따라 기울기가 크게 작아지므로 $sigmoid$보다는 양호하지만 gradient vanishing은 피할 수 없다.

3. Overfitting

3.1. Bias와 Variance

bias는 딥러닝 모델을 통해 얻은 예측값과 실제 정답과의 차이의 평균을 나타낸다. bias가 높으면 그만큼 예측값과 정답값 간의 차이가 크다고 할 수 있다. 한편 variance는 다양한 데이터들(data set)에 대하여 예측값이 얼만큼 변화할 수 있는지에 대한 양의 개념이며 딥러닝 모델이 얼만큼 유연성을 가지는지에 대한 의미로도 사용하며 variance의 원래 의미와 같이 예측값이 얼마나 퍼져 있어서 다양하게 출력될 수 있는지도 알 수 있다.



[그림 7] Bias와 Variance

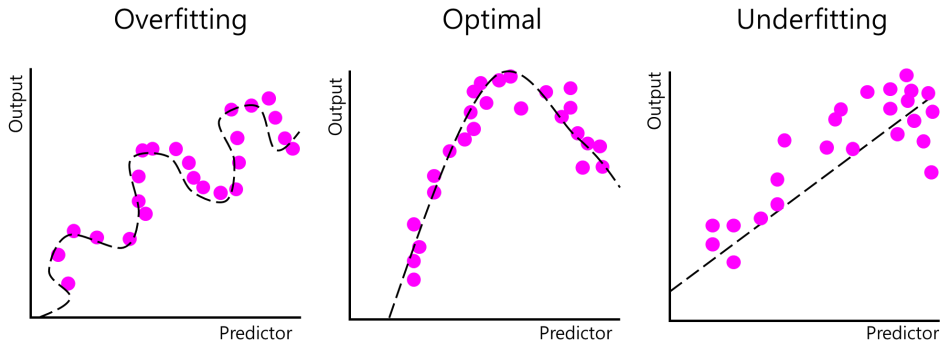
[그림 7]은 bias와 variance의 크고 작음에 따라 4가지 경우로 분류하고 딥러닝 모델이 예측한 값들은 핑크로, 정답 데이터의 위치는 초록이라고 했을 때 가장 적합한 구간은 예측값들이 정답 근방에 분포되어 있는 Low Bias & Low Variance이다.

3.2. Overfitting과 Underfitting

Supervised Learning의 목표는 주어진 데이터(training data)를 이용해 데이터를 잘 설명하는 모델을 만들고 만들어진 모델을 이용하여 학습하지 않은 데이터(test data)에서도 예측력이 우수한 모델을 만드는 것이 목표이다.

3.2.1. Overfitting과 Underfitting이란?

overfitting이란 training data에 딥러닝 모델이 너무 적합하게 학습되어 있어서 test data가 들어오게 되면 정확도가 떨어지는 현상을 말한다. 반면 training data도 학습을 하지 못한 상태는 underfitting이다. underfitting이 일어나는 이유는 학습 반복 횟수가 너무 적고 데이터들의 특성에 비해 딥러닝 모델이 너무 간단하거나 데이터 양이 너무 적은 문제 등이다.



[그림 8] Overfitting과 Underfitting

3.2.2. Overfitting의 해결방법

overfitting은 딥러닝 모델의 성능을 떨어뜨리는 주요 원인이다.

overfitting을 해결하기 위한 방법으로

첫째,모델의 데이터 양을 늘리는 방법이 있다. 딥러닝 모델은 데이터 양이 적을수록 해당 데이터의 특징이나 오류까지 암기해버려서 overfitting이 될 가능성이 높기 때문이다.

둘째는 neural network의 복잡도는 hidden layer의 수 등으로 결정된다. 복잡도를 줄임으로써 overfitting을 해결할 수 있다.

셋째는 학습과정에서 neural network의 일부를 사용하지 않는 방법이다. 그러나 이 방법은 training data에만 사용하고 test data에는 사용하지 않는 것이 일반적이다.

넷째, 쓸데없는 변수를 제거해서 입력값의 수를 줄이는 것인데 의미있는 변수들을 남기기 위해 주로 출력값 직전의 hidden 노드수를 줄이는 방법이다.

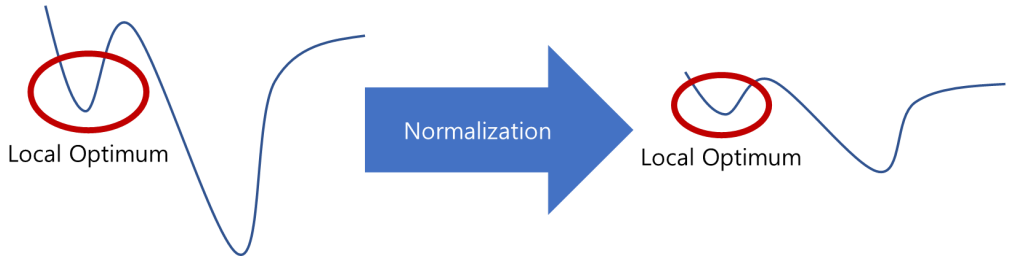
다섯째, 이 연구에서 구현을 통해 검증하고자 하는 batch normalization이 활용될 수 있다.

Ⅲ. 연구 방법

1. Batch Normalization 분석

1.1. Data Normalization

normalization을 하는 이유는 학습을 더 빨리 하기 위해서 또는 local optimum에 빠지는 문제를 해결하기 위해서 사용한다. [그림 9]에서 최솟값을 찾을 때 그래프를 전체적으로 이해하지 못하여 global optimum을 찾지 못하고 local optimum에 머물러 있게 된다. 이 때 normalization을 통해 local optimum에 빠질 가능성을 낮추어 줄 수 있다.



[그림 9] Data Normalization

1.2. Batch Normalization

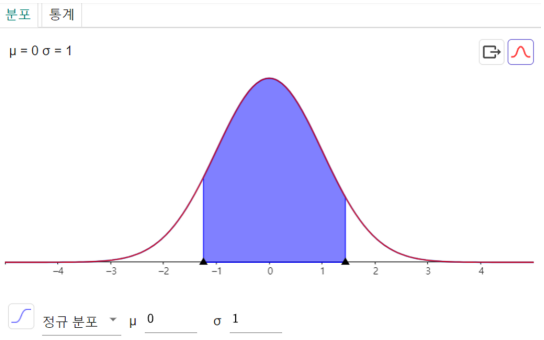
1.2.1. Batch

[그림 4]에서 gradient를 업데이트 하기 위해 모든 training data를 사용하게 된다. 이런 방식으로는 대용량의 데이터를 한번에 처리하지 못하기 때문에 데이터를 batch 단위로 나누어 학습을 하는 것이 일반적이다.

1.2.2. Batch Normalization

batch normalization은 학습과정에서 각 batch 단위 별로 다양한 분포를 가지더라도 각 batch별로 평균과 분산을 이용해 정규화 하는 것을 말한다. batch 단위나 레이어에 따라 입력값의 분포가 다르지만 정규화를 통해 zero mean gaussian(평균=0, 표준편차=1) 형태로 만들 수 있다.

batch normalization은 평균과 분산을 조정하는 함수를 따로 두는 것이 아니라 batch별로 계산해야 의미가 있다. 레이어가 많은 neural network일수록 같은 입력값을 가지더라도 가중치가 조금만 달라지면 완전히 다른 값을 얻을 수 있기 때문이다. batch normalization을 거치면 각 batch들이 표준 정규 분포의 형태로 만들 수 있다.



[그림 10] 표준 정규 분포 그래프

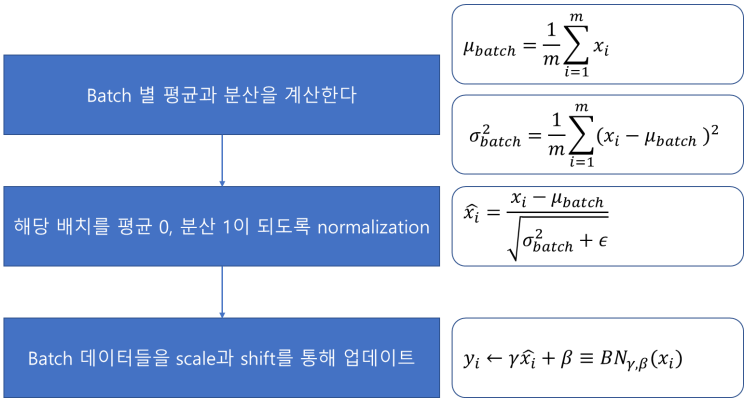
2. Batch Normalization를 통한 Overfitting 해결

batch normalizationdms 입력으로 들어오는 다양한 데이터 분포를 모두 정규화하여 모든 layer의 feature가 통일된 scale이 될 수 있고, 활성화함수에 맞게 적합한 분포로 변환할 수 있는 장점을 가진다.

또한, neural network에서 각 activation function의 미분값은 back-propagation 과정 속에서 계속 곱해지기 때문에 중요한 값이 된다. $sigmoid$ 함수의 경우 gradient vanishing 문제가 발생하게 되는데 이렇게 되면 가중치의 업데이트가 거의 일어나지 않아 속도가 느려지기 때문에 최적화에 실패하게 된다. 이를 해결하기 위한 방법으로 배치정규화를 활용한다.

3. Batch Normalization 알고리즘

3.1. Batch Normalization 알고리즘 분석



[그림 11] Batch Normalization 알고리즘($\gamma = scale, \beta = shift$)

normalization 이후 batch 데이터들은 scale(γ)와 shift(β)를 반영한 새로운 값으로 바뀌게 되는데 데이터를 계속 normalization하는 과정을 거치면 activation function의 비선형 성질을 잃게 되는 문제가 발생한다.

Ⅳ. 연구 결과

1. Batch Normalization 알고리즘을 python 언어로 구현

```
import numpy as np
def batchnorm_forward(x, gamma, beta, eps):
  # Inputs : shape
  N, D = x.shape



  #1: mean,difference
  sample_mean = 1./N * np.sum(x, axis = 0)
  x_centered = x - sample_mean


  #2: variance
  sq = x_centered ** 2
  var = 1./N * np.sum(sq, axis = 0)


  #3: standard deviation
  stddev = np.sqrt(var + eps)
  rstddev = 1./stddev


  #4: normalization
  x_norm = x_centered * rstddev


  #5: final output
  gammax = gamma * x_norm
```

```
    out = gammax + beta


    # caches
    cache = (x_norm,gamma,x_centered,rstddev,stddev,var,eps)


    return out, cache
```

2. 구현내용 설명

batch normalization 개념이 처음 발표된 논문(Ioffe & Szegedy, 2015, #) 상의 수학적 수식들을 Python 언어로 구현해보았다.

각 코드 라인에 대한한 설명을 풀이하자면,

```
def batchnorm_forward(x, gamma, beta, eps):
```

입력값 x는 training data set이고, gamma는 비선형적 특성을 보존하기 위한 scaling 값, beta는 bias역할을 하는 shift 값, eps는 계산할 때 0으로 나눠지는 문제(divided by 0)가 발생하는 것을 막기 위해 수치적 안정성을 보장하기 위한 아주 작은 숫자값이다.

```
 #Inputs : shape
  N, D = x.shape
```

training data set을 shape 타입 변수로 선언한다.

```
  #1: mean,difference
  sample_mean = 1./N * np.sum(x, axis = 0)
  x_centered = x - sample_mean
```

training data set의 평균값을 계산하고, 실제 입력값과 평균값의 차이를 구한다.

```
#2: variance
sq = x_centered ** 2
var = 1./N * np.sum(sq, axis = 0)
```

training data set의 분산을 계산한다.

```
#3: standard deviation
stddev = np.sqrt(var + eps)
rstddev = 1./stddev
```

계산된 분산을 이용하여 표준편차를 계산한다. 이 때 eps값은 계산할 때 0으로 나눠지는 문제가 발생하는 것을 막기 위한 수치적 안정성을 보장하기 위한 아주 작은 숫자이다.

```
#4: normalization
x_norm = x_centered * rstddev
```

x값의 정규화 계산한다.

```
#5: final output
gammax = gamma * x_norm
out = gammax + beta
```

비선형성을 보존하기 위해 scaling과 shift변환을 수행하여 최종 출력값 계산한다.

V. 결론 및 제언

1. 결론

본 연구에서는 딥러닝 학습 시 overfitting 문제를 분석하고 그 해결책으로 batch normalization에 대한 수학적 수식들을 python으로 구현해보았다. 딥러닝 관련 개념을 연구하며 발생할 수 있는 문제에 대해 관련 개념들을 이해하고 해결책을 프로그래밍 언어로 구현하여 복잡한 딥러닝 전체 학습 과정의 이해를 도울 수 있었다.

2. 제언

본 연구에서는 학습단계의 batch normalization에 대해서만 고려하였지만, 실제 테스트를 수행한 추론단계의 batch normalization의 알고리즘은 학습단계와는 유사하므로 추후 추가적인 구현 검증을 통해 분석하고자 한다.

참 고 문 헌

마사노리, 아. (2020). 딥러닝을 위한 수학*(*위키북스 데이터 사이언스 시리즈 *52)*. 위키북스.

한땀컴비, 6. 외. (2023). 한땀한땀 딥러닝 컴퓨터 비전 백과사전. 한땀컴비 외 6명. https://wikidocs.net/book/6651

Ioffe, S., & Szegedy, C. (2015, 7). Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift. *International conference on machine learning*, 448-456. https://doi.org/10.48550/arXiv.1502.03167