딥러닝 중간과제

인하대학교 산업경영공학과 12190625 배기웅

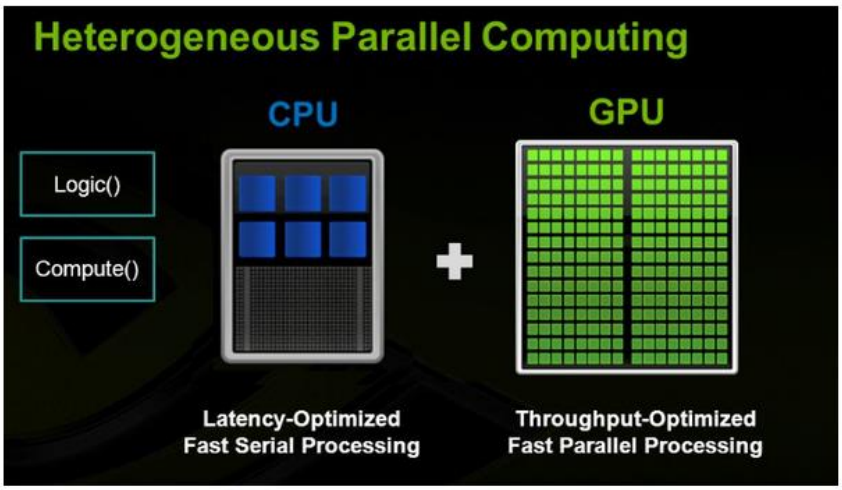
1. 딥러닝의 시대가 도래한 이유

'딥러닝'이라는 용어가 본격적으로 사용된 것은 2000 년대부터다. 하지만 그 근간이 되는 기법들은, 역전파는 1989 년, CNN 도 1989 년, 자연어 처리에 주로 사용되는 LSTM 은 1997 년에 개발되었다. 그렇지만 그 이후로 큰 발전이 없다가 2012 년도에 갑자기 딥러닝은 큰 부상을 하게 되었다. 왜 이렇게 딥러닝의 시대가 도래하게 되었는지, 왜 갑자기 딥러닝이 큰 이슈가 되었는지는 크게 세 가지 관점에서 바라볼 수 있다.

1.1 하드웨어의 발전

현재 시중에 판매되고 있는 CPU 는 20 세기 말과 비교했을 때 거의 5000 배가 빨라졌다. 그럼에도 불구하고 일반적인 딥러닝 모델은 이러한 성능의 10 배 이상의 계산능력을 필요로 한다.

그러던 중, 2000 년 대에 NVIDIA 와 AMD 와 같은 게임 회사들이 비디오 게임의 그래픽 성능을 높이기 위해 대용량 고속 병렬 칩(GPU)를 개발하는데 수십 억 달러를 투자하였다. 2007 년 NVIDIA 가 자사의 GPU 제품을 위한 프로그래밍 인터페이스인 CUDA 를 출시하게 되었다. 이로 인해 물리 모델링을 시작으로 다양한 병렬 애플리케이션의 대형 CPU 클러스터가 소량의 GPU 로 대체되기 시작했고, 계산량이 많은 행렬 곱셈도 빠르게 해결할 수 있게 되었다.

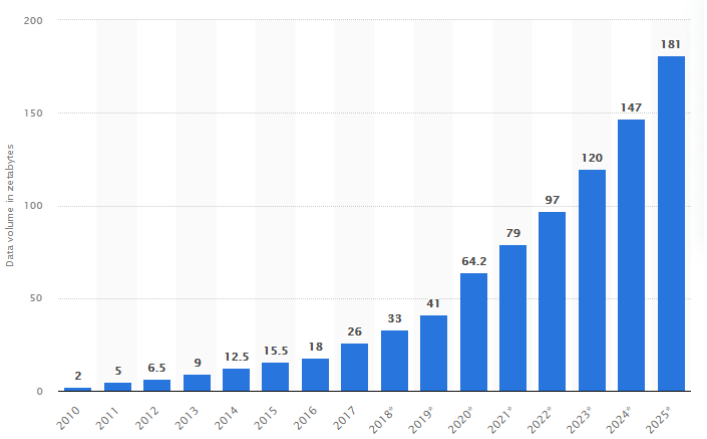


GPU 의 병렬 연산

(출처 : 엔비디아)

1.2 데이터

현재는 4 차 산업혁명이 계속 진행되고 있고, 그 가운데 AI 는 중요한 역할을 하고 있다. 딥러닝을 이 혁명의 증기 기관이라 한다면, 데이터는 이 기관을 움직이게 하는 연료와 같은 역할을 한다. 기관을 움직이는 연료가 없다면 어떤 것도 할 수 없듯이, 딥러닝도 데이터가 없다면 할 수 있는 것이 없다.



데이터의 용량 변화(단위 : 제타바이트)

(출처 : https://www.statista.com/statistics/871513/worldwide-data-created/)

기존에 있던 데이터들이 정형화된 데이터뿐 이었다면, 위에서 언급한 하드웨어의 발전과 함께 기존의 아날로그 데이터들이 디지털화 되었고, 기존엔 디지털로 다룰 수 없던 비정형 데이터 또한 데이터화 되어가며, 새롭게 등장하는 디지털 플랫폼들과 함께 인류는 유래 없는 양의 데이터를 축적해 나아가고 있다. 이런 정보들은 기존 데이터베이스 관리도구의 능력을 넘어서는 방대한 양이 되었으며, 이것을 '빅데이터(Big Data)'라 한다. 이러한 빅데이터 시대의 등장이 중요한 것은, 적은 양의 정보로도 판단을 내릴 수 있는 사람과는 다르게 현재의 딥러닝 모델들을 실행시키기 위해서는 많은 양의 데이터를 필요로 하며, 이와 같은 빅데이터의 시대는 딥러닝을 구현하는 환경을 조성하기 충분한 요건을 갖춰 줄 수 있게 되었다.

1.3 알고리즘

인공지능에 대한 개념은 1950 년대부터 논의되어 왔지만, XOR 문제라는 첫번째 겨울, 고가의 비용이라는 두번째 겨울을 맞이하며 2000 년대 후반에는 층이 깊은 복잡한 형태의 신경망을 학습시킬 수 있는 적합한 방법을 찾지 못하였다. 단적인 예로 층의 깊이가 깊어졌을 때 발생하는 기울기 소실등의 문제로, 얕은 형태의 신경망만을 이용하였고, 이러한 얕은 신경망은 SVM 과 랜덤 포레스트와 같은 머신러닝 기법들에 비해 성능이 뛰어나지 못해 딥러닝은 빛을 보지 못하였다. 신경망을 훈련하기 위한 피드백 신호가 층이 늘어남에 따라 값이 희미해지기 때문이다. 하지만, 하드웨어도 발전하고, 이용할 수 있는 데이터가 점차 늘어나며 인공지능의 두번째 겨울, 고가의 비용이라는 문제를 어느정도 해결할 수 있게 되었고, 더 많은 연구자들이 딥러닝에 관련된 연구에 참여하며 2010 년경부터 몇 가지 알고리즘들이 탄생하게 되면서 다시 한번 부흥기를 맞이하게 되었다.

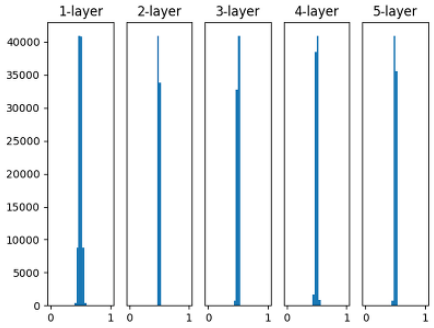
1.3.1 활성화함수

활성화함수란 입력된 데이터의 가중 합을 출력신호로 변환하는 함수이다. 밑의 3 장 활성화함수에 대해 소개할 때 자세히 설명하였다.

1.3.2 가중치 초기화 방법

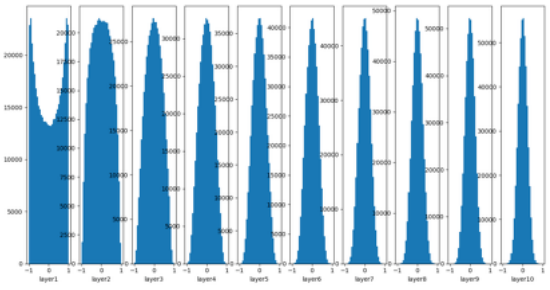
초기 가중치 값들을 표준 정규분포를 이용해 초기화 시켰을 때 Sigmoid 함수의 출력 값이 0 과 1 에 치우치게 되는 현상이 발생하게 된다. 만약 표준편차가 1 보다 더 커지게 되면, 즉 가중치의 값들이 0 에서 점점 더 멀어질수록 Sigmoid 출력값이 0 과 1 이 되는 현상은 더 심해지게 된다. 이렇게 되면 그래디언트 값이 0 에 가까운 값을 갖게 되는 현상이 더 심해지게 되고, 이는 Gradient Vanishing 현상의 원인이 된다.

이러한 현상은 표준편차를 줄이는 방법을 통해 해결하였다.



위의 그래프는 표준편차를 0.01 로 하는 정규분포 형태로 가중치를 초기화 시켰을 때의 sigmoid 함수 출력값의 분포이다. 표준편차가 1 일 때는 0 과 1 에 몰려 있었지만, 지금 이 경우에는 0.5 에 몰려있음을 볼 수 있다. 이렇게 구성이 된다면 표준편차가 1 일 때보다 의미 있는 그래디언트 값을 얻을 수 있어 Gradient Vanishing 현상을 어느 정도 해결할 수 있다.

가중치 초기화 방법은 단순히 표준편차의 값을 작게 하는 방법을 기반으로 하여 Xavier Initialization 방법으로 발전하게 된다. Xavier 초기화 방법은 표준 정규분포를 입력 개수의 표준 편차로 나누어주는 방식이다.



Xavier 초기화 방법을 사용한 결과, 층이 10 층까지 깊어 졌음에도 표준 정규분포 형태로 잘 출력되고 있음을 알 수 있다.

Xavier 초기화 방법은 Sigmoid 함수에 적절한 초기화 방법이다. 합성곱 연산에 주로 사용되는 ReLU 함수를 사용할 때는 주로 He 초기화 방법을 사용한다. He 초기화 방법은 인풋 개수의 절반의 제곱근으로 나누어 정규화 시키는 방법이다.

1.3.3 최적화 방법

딥러닝에서 최적화 방법이란 학습 데이터셋을 이용하여 모델을 학습할 때 데이터의 실제 결과와 모델이 예측한 결과를 기반으로 잘 줄일 수 있도록 하는 것을 말한다. 자세한 내용은 4.2.4 장에서 설명하였다.

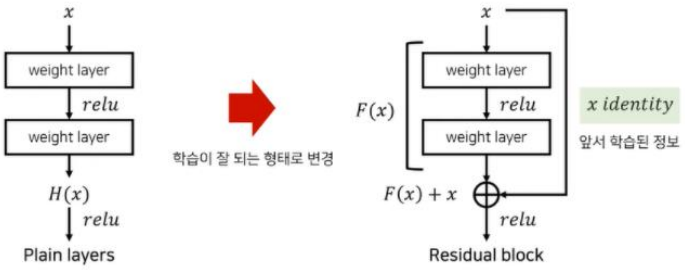
1.3.4 배치 정규화

배치 정규화는 학습의 효율을 높이기 위해 도입되었다. 배치 정규화는 활성화 함수의 활성화 값 또는 출력 값을 정규분포로 만들어주는 작업을 한다. 각 은닉층에서 정규화를 실행하면서 입력 분포를 일정하게 만들어주어 학습 속도가 개선된다는 장점이 있다.

배치 정규화를 하는 가장 큰 목적은 바로 입력 분포의 균일화이다. 딥러닝 학습은 이전 층에서 온 값에 가중치를 곱하는 과정이기 때문에 학습이 계속되면 입력분포가 변화하면서 가중치가 엉뚱한 방향으로 갱신될 수 있다. 이러한 현상을 방지하기 위해 중간중간에 배치 정규화를 한다.

1.3.5 잔차 연결 (Residual Connection)

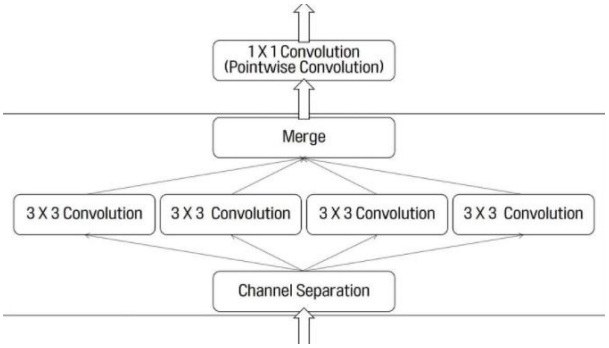
잔차 연결이란, input 은 그대로 가져오고 나머지 잔여 정보인 F(x)만 추가적으로 더해주는 단순한 형태로 만들어주는 연결을 말한다.



전체를 학습하는 것이 아닌, 추가적인 정보만 학습하면 되므로 학습이 쉬워지고 수렴을 더 잘 할 수 있다는 장점이 있다. 잔차 연결은 구현이 간단하며, 학습 난이도가 낮아지고 깊이가 깊어질수록 높은 정확도 향상을 보인다는 특징이 있다.

1.3.6 깊이별 분리 합성곱

깊이별 분리 합성곱이란 입력 채널 별로 채널을 분리해서 따로따로 합성곱 연산을 수행하는 방법이다.



2. 신경망이 작동하는 이유는 무엇인가? 단순한 행렬 계산만으로 원하는 답이 나오는 이유는 무엇인가?

2.1 신경망 작동의 목적

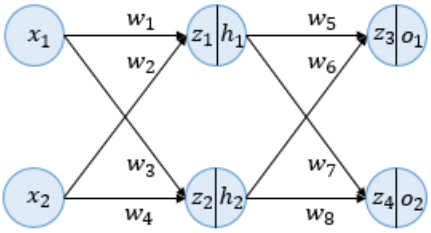
우선 신경망이 작동하는 목적에 대해서 알아야 한다. 신경망은 크게 입력층, 은닉층, 출력층 이렇게 세 부분으로 구분할 수 있는데, 은닉층은 입력층으로부터 들어오는 입력값에 적절한 파라미터를 곱한 뒤, 출력층에서 결과 값을 도출하여 실제값과 최대한 비슷하게 출력할 수 있도록 역전파를 하여 파라미터를 조정한다. 즉, 적절한 파라미터를 생산하는 것이 신경망의 목적이라고 할 수 있고, 적절한 파라미터를 만들기 위해 신경망이 작동한다고 할 수 있다.

2.2. 신경망 작동 원리

그렇다면 어떻게 신경망은 파라미터를 조절하여 결과값을 출력하는가? 크게 가중치를 곱하여 결과값을 출력하는 순전파 과정과 가중치를 조절하는 역전파과정으로 나뉜다. 순전파(Forward propagation)는 신경망 모델의 입력층부터 출력층까지 순서대로 변수들을 계산하고 저장하는 것을 의미한다. 역전파(Backpropagation)는 신경망 파라미터들에 대한 그래디언트(gradient)를 계산하는 방법을 의미한다. 단순히 정의로 이해하기 보다는 예제를 이용한 계산 과정을 통해 알아보도록 한다.

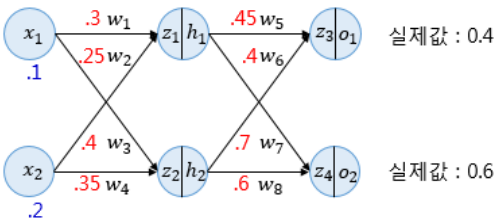
2.2.1 인공 신경망의 이해

예제에 사용될 인공 신경망은 아래 그림과 같다.



계산을 보다 편리하게 하기 위해 입력층 1 개, 은닉층 1 개, 출력층 1 개를 가진다고 가정한다. 해당 인공 신경망은 두 개의 입력과 두 개의 은닉층 뉴런, 그리고 두 개의 출력층 뉴런을 사용한다. 그리고 은닉층과 출력층 노드에서 사용할 활성화함수는 시그모이드 함수를 사용한다. 해당 인공 신경망은 편향(bias)은 고려하지 않는다.

2.2.2 순전파(Forward Propagation)



파란색 숫자는 입력값을 의미하고, 빨간색 숫자는 각 가중치의 값을 의미한다.

각 입력은 입력층에서 은닉층 방향으로 향하면서 각 입력에 해당하는 가중치와 곱해지고, 결과적으로 가중합으로 계산되어 은닉층 뉴런의 시그모이드 함수의 입력값이 된다. z_1과 z_2는 시그모이드 함수의 입력으로 사용되는 각각의 값에 해당된다.

$$z_1 = w_1x_1 + w_2x_2 = 0.3 * 0.1 + 0.25 * 0.2 = 0.08$$

$$z_2 = w_3x_1 + w_4x_2 = 0.4 * 0.1 + 0.35 * 0.2 = 0.11$$

z_1과 z_2는 각각의 은닉층 뉴런에서 시그모이드 함수를 지나게 되는데, 시그모이드 함수가 출력하는 값은 은닉층 뉴런의 최종 출력값(h_1과 h_2)이 된다.

$$h_1 = sigmoid(z_1) = 0.51998934$$

$$h_2 = sigmoid(z_2) = 0.52747230$$

h_1과 h_2 이 두 값은 다시 출력층의 뉴런으로 향하게 되는데, 이때 다시 각각의 값에 해당되는 가중치와 곱해지고, 다시 가중합되어 출력층 뉴런의 시그모이드 함수의 입력값(z_1과 z_2)이 된다.

$$z_3 = w_5h_1 + w_6h_2 = 0.45 * h_1 + 0.4 * h_2 = 0.44498412$$

$$z_4 = w_7h_1 + w_8h_2 = 0.7 * h_1 + 0.6 * h_2 = 0.68047592$$

z_3과 z_4가 출력층 뉴런에서 시그모이드 함수를 지난 값은 이 인공 신경망이 최종적을 계산한 출력값이다. 실제값을 예측하기 위한 값으로서 예측값이라고도 부른다.

$$\sigma_1 = sigmoid(z_3) = 0.60944600$$

$$\sigma_2 = sigmoid(z_4) = 0.66384491$$

이렇게 예측값을 구하고, 이 예측값을 바탕으로 실제값과의 차이인 오차값을 구한다. 손실함수로는 평균제곱오차(MSE)를 사용한다.

$$E_{\sigma_1} = \frac{1}{2}\left(target_{\sigma_1} - output_{\sigma_1}\right)^2 = 0.02193381$$

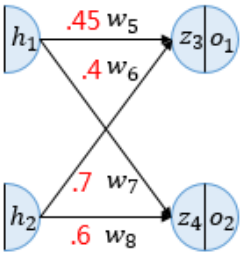
$$E_{\sigma_2} = \frac{1}{2}\left(target_{\sigma_2} - output_{\sigma_2}\right)^2 = 0.00203809$$

$$E_{total} = E_{\sigma_1} + E_{\sigma_2} = 0.02397190$$

위의 계산 결과 전체 에러값은 0.02397190 이다.

2.2.3 역전파 1 단계

순전파가 입력층에서 출력층으로 향한다면, 역전파는 반대로 출력층에서 입력층 방향으로 계산하면서 가중치를 업데이트한다. 출력층 바로 이전의 은닉층을 N 층이라고 했을 때, 출력층과 N 층 사이의 가중치를 업데이트하는 단계를 역전파 1 단계, 그리고 N 층과 N 층의 이전 층 사이의 가중치를 업데이트하는 단계를 역전파 2 단계라고 한다.



역전파 1 단계에서 업데이트해야 하는 가중치는 w_5, w_6, w_7, w_8 이다. 가중치가 업데이트되는 원리는 동일하게 적용되므로, w_5로 가중치를 갱신한다. 가중치 w_5를 업데이트하기 위해선 $\frac{\partial E_{total}}{\partial w_5}$을 계산해야 한다.

$\frac{\partial E_{total}}{\partial w_5}$는 미분 연쇄법칙 (Chain Rule)을 이용해서 다음과 같이 풀어 쓸 수 있다.

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial z_3}\frac{\partial z_3}{\partial w_5}$$

위의 식에서 우변의 세 개의 각 항에 대해서 순서대로 계산해본다. 우선 첫 번째 항이다.

$$E_{total} = E_{\sigma_1} + E_{\sigma_2} = \frac{1}{2}\left(target_{\sigma_1} - output_{\sigma_1}\right)^2 + \frac{1}{2}\left(target_{\sigma_2} - output_{\sigma_2}\right)^2$$

$\frac{\partial E_{total}}{\partial \sigma_1}$는 다음과 같다.

$$\frac{\partial E_{total}}{\partial \sigma_1} = 2 * \frac{1}{2}\left(target_{\sigma_1} - output_{\sigma_1}\right)^{2-1} * (-1) + 0 = -\left(target_{\sigma_1} - output_{\sigma_1}\right) = 0.20944600$$

두 번째 항 계산은 다음과 같이 한다. σ_1이라는 값은 시그모이드 함수의 출력값이다. 그런데 시그모이드 함수의 미분은 $f(x) * \left(1 - f(x)\right)$이다. 따라서 다음과 같이 계산하면 된다.

$$\frac{\sigma_1}{z_3} = \sigma_1 * (1 - \sigma_1) = 0.60944600 * (1 - 0.60944600) = 0.23802157$$

마지막으로 세 번째 항은 h_1과 동일하다.

$$\frac{z_3}{w_5} = h_1 = 0.51998934$$

우변의 모든 항을 계산하였고, 각 항들을 곱해서 $\frac{\partial E_{total}}{\partial w_5}$를 구한다.

$$\frac{\partial E_{total}}{\partial w_5} = 0.20944600 * 0.23802157 * 0.51998934 = 0.02592286$$

이제 이렇게 계산한 이 값을 경사하강법에 사용하여 가중치를 업데이트한다. 이때 학습률 α는 0.5 라고 가정한다.

$$w_5^+ = w_5 - \alpha\frac{\partial E_{total}}{\partial w_5} = 0.45 - 0.5 * 0.02592286 = 0.43703857$$

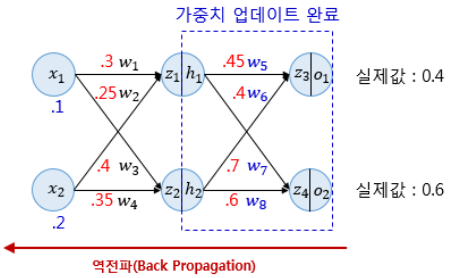
위와 같은 방식으로 w_6^+, w_7^+, w_8^+ 을 계산하면 다음과 같이 나온다.

$$w_6^+ = 0.38685305$$

$$w_7^+ = 0.69629578$$

$$w_8^+ = 0.59624247$$

2.2.4 역전파 2 단계



이제 입력층 방향으로 이동하며 다시 계산을 한다. 이번에 계산할 가중치는 w_1, w_2, w_3, w_4 이다. 이 가중치도 위에서 한 것과 같이 미분의 연쇄 법칙(Chain Rule)을 통해 계산할 수 있고 다음과 같은 결과를 얻을 수 있다.

$$w_1^+ = w_1 - \alpha \frac{\partial E_{total}}{\partial w_1} = 0.1 \; - \; 0.5 * 0.00080888 \; = \; 0.29959556$$

$$w_2^+ = w_2 - \alpha \frac{\partial E_{total}}{\partial w_2} = 0.24919112$$

$$w_3^+ = w_3 - \alpha \frac{\partial E_{total}}{\partial w_3} = 0.39964496$$

$$w_4^+ = w_4 - \alpha \frac{\partial E_{total}}{\partial w_4} = 0.34928991$$

2.2.5 순전파 및 결과 확인

이렇게 갱신한 가중치 값을 이용하여 또 다시 순전파 과정을 진행하여 오차 값을 구한다.

2.2.2 에서 진행한 순전파 과정을 또 다시 진행한 결과, 오차값은 다음과 같이 나온다.

$$E_{\sigma_1} = \frac{1}{2}\left(target_{\sigma_1} - output_{\sigma_1}\right)^2 = 0.02125445$$

$$E_{\sigma_2} = \frac{1}{2}\left(target_{\sigma_2} - output_{\sigma_2}\right)^2 = 0.00198189$$

$$E_{total} = E_{\sigma_1} + E_{\sigma_2} = 0.02323634$$

기존의 오차값은 0.023978190 이었음을 보아 역전파 과정으로 오차값이 감소한 것을 확인할 수 있다. 이렇게 인공 신경망의 학습은 오차를 최소화하는 가중치를 찾는 목적으로 순전파 과정과 역전파 과정을 반복한다.

2.3 행렬 연산의 장점

위 예제에서 언급한 인공 신경망은 입력 노드, 은닉층의 노드, 출력 노드가 각각 2 개이고, 은닉층의 개수가 1 개인 간단한 구조의 신경망이다. 신경망 구조가 간단함에도 불구하고 계산량이 적지 않다. 하지만 여기서 노드의 개수나 신경망의 깊이가 깊어지게 되면 연산과정이 상당히 복잡해진다. 이럴 때 행렬 연산을 이용한 병렬 계산처리 방법을 이용하면 보다 더 쉽고 빠르게 계산할 수 있다.

행렬이란 수 또는 다항식 등을 직사각형 모양으로 배열한 것이다. 딥러닝에서 행렬은 복잡한 식들을 행렬로 간단히 표기할 수 있다는 장점을 가진다.

2.4 결론

신경망이 작동하는 이유는 입력값에 대한 적절한 파라미터를 찾기 위해서이다. 그런데 실생활에서 주로 사용한 신경망은 깊이가 깊고 노드의 개수도 많기 때문에 대량의 복잡한 계산과정이 필요하다.

하지만 행렬을 이용한 병렬 계산과정을 통해 대체 가능하기 때문에, 단순한 행렬 계산으로 가중치를 곱하고 갱신하는 학습과정으로 신경망이 작동되어 우리가 원하는 답인 실제값에 근접한 출력값을 얻을 수 있다.

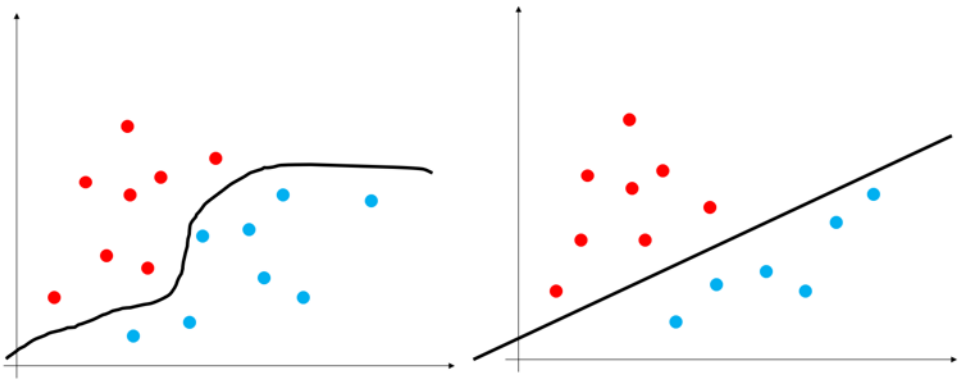
3. 활성화 함수를 사용하는 이유와 활성화 함수의 종류

3.1 왜 비선형 함수를 사용해야 하는가?

활성화함수로 주로 비선형 함수를 사용한다. 만약 활성화함수를 선형 함수로 사용하게 되면 다중 신경망을 구성하는 이유가 없기 때문이다.

예를 들어 $f(x) = ax, g(x) = bx$라고 할 때, f(x)와 g(x), 두 함수로 이루어진 2중 신경망이 있다고 하자. 이 2중 신경망과 $h(x) = abx$인 단일 신경망은 최종 출력값에 차이가 없어 복잡하게 신경망을 구성할 수 없다.

또, 선형 함수를 사용하면 선형 분리 문제만 해결할 수 있다. 아래 그림과 같이 A그룹과 B그룹이 있고 이를 분리하려 한다.

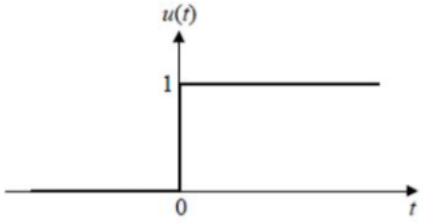


오른쪽과 같은 경우는 선형함수로 쉽게 분류가 가능하지만 왼쪽과 같은 경우는 선형함수로는 절대 분류할 수가 없다. 이러한 이유들로 선형함수를 활성화함수로 사용하기에는 여러 한계점이 존재하기 때문에 비선형 함수를 사용한다.

딥러닝 모델에서 사용하는 비선형함수는 정말 다양하다.

3.2 활성화함수의 종류

3.2.1 단위 계단 함수(Step)

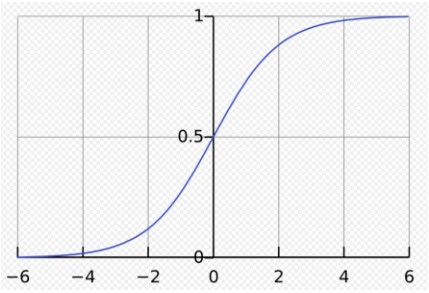


$$u(t) = \begin{cases} 0 & (t < 0) \\ 1 & (t > 0) \\ 0.5 & (t = 0) \end{cases}$$

단위 계단 함수 도는 헤비사이드 계단함수(Heaviside Step Function)

0보다 작은 실수에 대해서 0, 0보다 큰 실수에 대해서 1, 그리고 0에 대해서 0.5의 값을 갖는 함수이다.

3.2.2 Sigmoid 함수, 로지스틱 함수



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

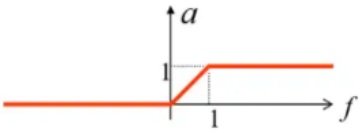
Sigmoid 함수는 주로 로지스틱 회귀분석에 사용이 되는 활성화함수이다.

> ※ *로지스틱 회귀분석(Logistic Regression Analysis)이란*
>
> 로짓분석이라고도 불리며, 어떤 사건이 발생할지에 대한 직접 예측이 아닌, 그 사건이 발생할 확률을 예측하는 분석방법이다. 로지스틱 회귀모형은 반응변수가 범주형(이항/다항)이며, 일반화 선형 모형의 특수한 형태로 S형 곡선을 그리는 함수모형이다.

정의역은 실수 전체이고, 치역은 0과 1사이의 값이다. 0.5를 기준으로 0.5보다 크면 1로 판단하고, 작으면 0으로 간주한다.

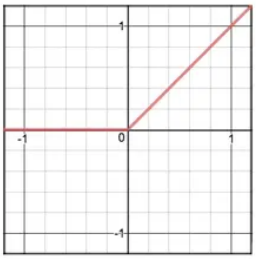
정의역과 비교했을 때 치역은 매우 작게 매핑이 된다. 이로 인해 입력값에 큰 변화가 생기더라도 output에서는 작은 변화를 보이게 되는데 이렇게 입력값이 커질수록 기울기가 0에 수렴하는 현상을 기울기 소멸문제(Gradient Vanishing Problem)라고 한다. 이러한 문제를 해결하기 위해 값을 짓이겨 넣는 squishing 방식을 갖지 않는 활성화함수인 ReLU 활성화 함수를 주로 사용한다.

3.2.3 Ramp함수



Sigmoid 함수와 매우 비슷한 형태로, 0과 1 사이의 값을 출력 값으로 가진다. Sigmoid는 부드러운 곡선 형태의 logistic 회귀 모형이었다면, Ramp함수는 선형 함수모형이라는 특징이 있다.

3.2.4 ReLU (Rectified Linear Unit)

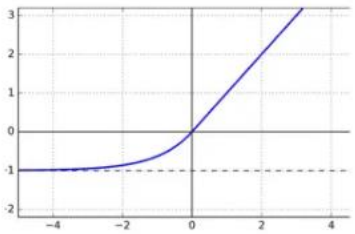


$$ReLU(x) = max(0, x)$$

입력 값이 0보다 작으면 0을, 0보다 크면 입력값 그대로 출력하는 활성화함수이다. 주로 Convolution 연산에 사용이 된다. Sigmoid 함수의 단점인 Gradient Vanishing 현상을 해결하기 위해 도입되었으며, 단순하지만 성능이 좋아 사용된다.

하지만 입력값이 0이하인 경우 다음 층에서 활성화되지 않는다는 Dying ReLU 문제가 발생한다는 특징이 있다.

3.2.5 ELU함수(Exponential Linear Unit)

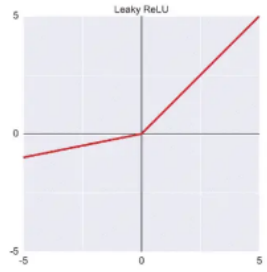


$$ELU(x) = \begin{cases} x & , \quad x > 0 \\ a(e^x - 1), & x \leq 0 \end{cases}$$

ReLU함수는 값이 0 이하일 때는 0으로 변환을 하기 때문에 뉴런이 0을 출력하여 더 이상 학습이 되지 않는다는 문제(Dying ReLU Problem)가 발생한다. 이러한 문제점을 해결하고자 ELU함수가 탄생하였다

ReLU의 모든 장점을 포함하면서 Dying ReLU 문제를 해결하고, 출력값이 거의 zero-centered에 가깝다는 특징이 있지만 입력값이 0보다 작을 때 exp함수 계산비용이 발생한다는 단점이 있다.

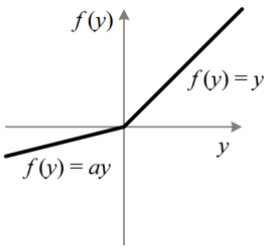
3.2.6 LeakyReLU



$$ELU(x) = \begin{cases} x & , \quad x > 0 \\ 0.01x, & \quad x \leq 0 \end{cases}$$

LeakyReLU 함수도 ELU와 같이 Dying ReLU의 문제점을 해결하기 위해 만들어졌다. 입력값이 0보다 큰 경우에는 x값을 그대로 가진다는 점에서는 ReLU함수, ELU함수와 공통점을 가지지만, 0미만일 때 ReLU함수는 0, ELU함수는 비선형 함수를 가지는 반면 LeakyReLU함수는 선형 함수를 가진다는 차이점이 있어 연산 또한 빠르게 진행할 수 있다.

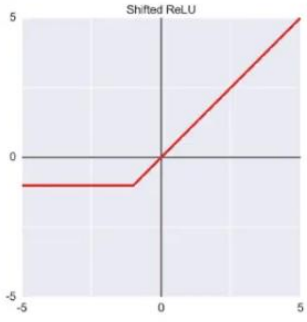
3.2.7 PReLU 함수(Parametric ReLU)



$$ELU(x) = \begin{cases} x & , \quad x > 0 \\ a\,x, & \quad x \leq 0 \end{cases}$$

Leaky ReLU처럼 0보다 작을 때 0이 아닌 기울기를 가지는 선형함수를 가지지만, 기울기가 고정되어 있지 않다. 기울기 a는 각 레이어를 지날수록 알맞게 업데이트를 시켜 학습시킨다.

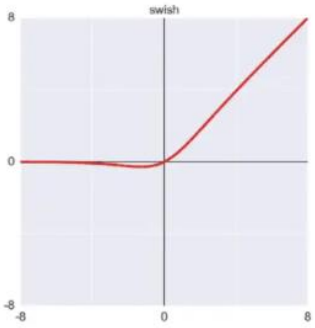
3.2.8 Shifted ReLU 함수



$$f(x) = max(0, \Delta)$$

기존의 ReLU함수를 약간 이동시킨 함수이다. 수평 및 수직 이동을 할 수 있어 유동적이다.

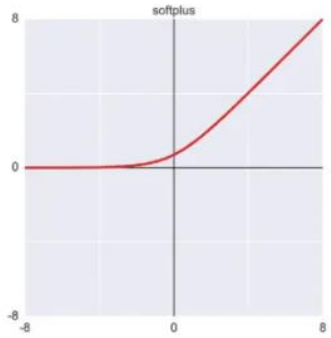
3.2.9 Swish 함수



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

ReLU함수와 Sigmoid의 특성을 합친 활성화함수이다. 0 바로 왼쪽에 부드러운 범프가 있다는 것이 특징이다. 약간의 범프를 추가함으로써 ReLU보다 더 효율적으로 계산할 수 있다.

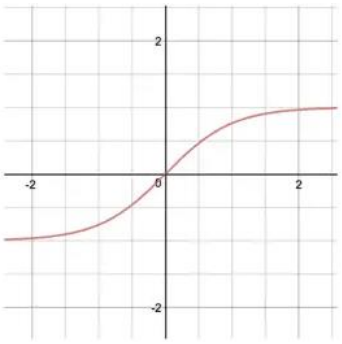
3.2.10 Softplus 함수



$$softplus(x) = ln(1 + e^{x})$$

ReLU함수와 유사하지만 0부분이 매끄럽고 미분 가능하다. 2001년에 처음 도입되었으며, dying ReLU 문제를 해결할 수 있다. sigmoid함수나 tanh함수는 (0,1), (-1,1) 등 일부 범위 안에서 값을 출력하지만, softplus 함수는 (0,∞)에서 값을 출력할 수 있다.

3.2.11 Tanh 함수

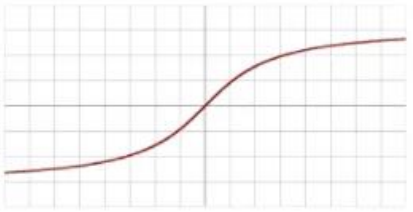


$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

하이퍼볼릭 탄젠트함수라고 말한다. 하이퍼볼릭함수는 쌍곡선 함수라고 하며, 표준 쌍곡선을 매개변수로 표시할 때 나온다.

하이퍼볼릭 탄젠트 함수는 중앙값이 0이기 때문에 경사하강법 사용 시 Sigmoid함수에서 발생하는 편향 이동이 발생하지 않는다. 즉, 기울기가 양수, 음수 모두 나올 수 있기 때문에 sigmoid함수보다 학습 효율성이 뛰어나다. 또한 치역이 -1에서 1사이로 sigmoid함수의 치역보다 범위가 넓기 때문에 출력값의 변화 폭이 넓고 이로 인해 기울기 소실(Gradient Vanishing) 문제가 덜 발생한다는 장점이 있다.

3.2.12 arctangent 함수



$$f(x) = \frac{\pi}{2} tan^{-1}(x)$$

Arctangent함수는 sigmoid함수, tanh함수와 유사하며 -2에서 2 사이의 출력값을 가진다.

3.2.13 Softmax 함수

softmax함수는 세 개 이상으로 분류하는 다중분류(multi-classification) 문제에서 사용되는 활성화 함수이다. softmax함수가 분류하는 클래스가 n개라고 할 때, n차원의 벡터를 입력 받아 각 클래스에 속할 확률을 추정한다.

$$y_k = \frac{e^{a_k}}{\Sigma e^{a_i}}$$

4. Deep Learning 에서 매개변수는 어떤 것들이 있는가, 또 어떻게 설정해야 하는가?

4.1 parameter 와 hyperparameter 의 차이

딥러닝 학습과정에서 두 종류의 변수가 사용되는데, parameter 와 hyperparameter 이다. Parameter 는 모델 내부에서 결정되는 변수로, 가중치인 w 와 편향인 b 가 있다. Hyperparameter 는 사용자가 조정할 수 있는 변수로 초매개변수라고도 말한다.

4.2 Hyperparameter

4.2.1 학습률 (learning rate, lr)

기울기 방향으로 얼마나 빠르게 이동할 것인지를 결정하는 변수이다. 모델을 학습시킬 때 적절한 학습률을 설정하는 것이 매우 중요하다. 학습률이 너무 작으면 학습의 속도가 너무 크고, 학습률이 너무 크게 되면 최솟값에 수렴하기 어렵다.

4.2.2 Batch size, Epoch, Iteration

Batch size 는 Batch set 수행을 위해 전체 학습 데이터를 등분하는 크기를 말한다. 전체 Dataset 을 학습시키기엔 한계나 부담이 되기 때문에 Dataset 을 일정 크기로 나눠 학습을 시킨다. 이때 여기서 나눠진 일정 크기가 바로 Batch size 가 된다.

Epoch 는 학습을 반복하는 횟수이다. 학습을 많이 시키면, 즉 epoch 의 값을 높이면 정확도가 높아질 것이라고 생각할 수 있지만 너무 많이 하면 overfitting 현상이 발생할 수 있기 때문에 적절한 값으로 epoch 를 설정하는 것이 중요하다.

Iteration 은 Batch size 크기로 나눠진 데이터셋을 전부 다 학습하여 전체 Dataset 을 1 epoch 학습하는데 실행하는 횟수를 의미한다.

1 Epoch = Batchsize x Iteration

4.2.3 Loss Function

손실함수란 입력에 따른 예측값과 실제값의 차이를 계산하는 함수이다.

4.2.3.1 MSE

평균 제곱 오차, Mean Squared Error

$$MSE = \frac{\Sigma_{i=1}(y\_i - \overline{y\_i})2}{n}$$

평균 제곱 오차는 실제값과 예측값의 차이인 오차의 제곱에 대한 평균 값을 말한다. MSE 의 값이 작을수록 원본과의 오차가 작은 것이므로 높은 정확도를 의미한다.

4.2.3.2 MAE

평균 절대 오차, Mean Absolute Error

$$MSE = \frac{\Sigma_{i=1}|y\_i - \overline{y\_i}|}{n}$$

평균 절대 오차는 모든 절대 오차의 평균값이다. 실제값과 예측값의 차이인 오차를 이용한다는 점에서는 MSE 와 비슷하지만 제곱을 하지 않고 절댓값으로 오차를 표현한다는 차이점을 가진다.

이러한 MAE 와 MSE 의 차이는 Norm 의 관점에서도 설명할 수 있다. Norm 이란 벡터의 길이 혹은 크기를 측정하는 방법(함수)이다. 그리고 Norm 이 측정한 벡터의 크기는 원점에서 벡터 좌표까지의 거리 혹은 Magnitude 라고 한다.

Norm 은 크게 1 차 Norm 과 2 차 Norm 으로 구성이 된다. 1 차 Norm 은 맨허튼 노름(Manhattan Norm)이라고도 불리며, 벡터의 요소에 대한 절댓값의 합이다. 2 차 Norm 은 제곱을 한 것으로, n 차원 좌표 평면(유클리드 공간)에서의 벡터의 크기를 계산하기 때문에 유클리드 노름(Euclidean Norm)이라고도 한다.

아래 예제는 1 차 Norm 과 2 차 Norm 을 Python 으로 구현한 소스코드이다.

```
import numpy as np
x = np.random.randint(low  =1, high = 10, size = (5,3))
print(x)
```

```
[[9 9 1]
 [5 9 5]
 [8 4 3]
 [3 8 5]
 [5 3 4]]
```

```
L1_norm = np.linalg.norm(x, axis = 1, ord = 1)
L2_norm = np.linalg.norm(x, axis = 1, ord = 2)

print("1차 노름입니다 : ",L1_norm)
print("평균 :",sum(L1_norm)/x.shape[1])
print()
print("2차 노름입니다 : ",L2_norm)
print("평균 :",sum(L2_norm)/x.shape[1])
```

```
1차 노름입니다 :  [19. 19. 15. 16. 12.]
평균 : 27.0

2차 노름입니다 :  [12.76714533 11.44552314  9.43398113  9.89949494  7.07106781]
평균 : 16.872404119199015
```

4.2.3.3 RMSE

평균 제곱근 편차, 평균 제곱근 오차(Root Mean Squared Error)

$$RMSE = \sqrt{\frac{\Sigma_{i=1}(y_i - \overline{y_i})2}{n}}$$

RMSE 는 각각의 예측값과 실제값의 차이인 오차를 구하고, 그 오차들의 제곱의 평균값들에 대한 제곱근 값이다. RMSE 는 예측 대상의 크기에 따라 RMSE 수치의 크기 또한 영향을 받기 대문에 크기가 다른 타 예측모델과 RMSE 수치를 통한 비교가 불가능하다는 특징을 가진다.

4.2.3.4 binary_crossentropy

$$BCE = -\frac{\Sigma y_i log\big(h(x_i;\theta)\big) + (1-y_i) log\big(1-h(x_i;\theta)\big)}{n}$$

주로 이진분류(binary classification)문제에서 사용되는 손실함수이다.

4.2.3.5 categorical crossentropy

$$CE = -\Sigma t_i log(f(s)_i)$$

Softmax 활성화 함수 뒤에 cross-entropy loss 를 붙인 형태로 주로 사용하기 때문에 Softmax loss 라고도 불리며, 주로 다중 분류(Multi-Classification)문제에 사용되는 손실함수이다. MSE(Mean Squared Error)보다 CE Loss 가 더 빠르게 수렴한다는 특징이 있어 다중 분류 문제에서 클래스를 구분할 때 softmax 함수와 Cross Entropy 손실함수의 조합을 많이 사용한다고 한다.

4.2.3.6 Sparse Categorical Crossentropy

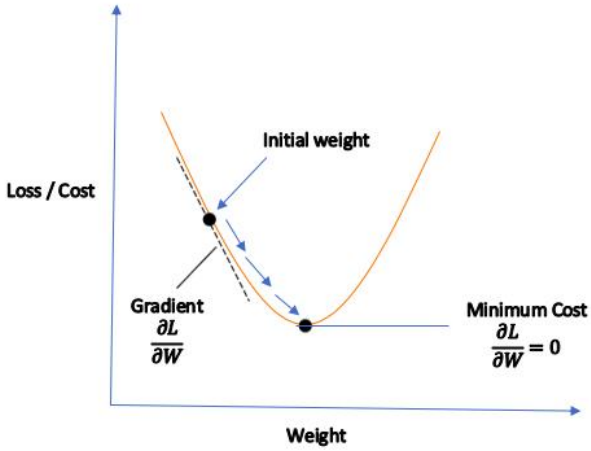
Sparse categorical crossentropy 도 다중 분류에 사용되는 손실함수이다. Sparse Categorical Crossentropy 는 라벨값이 0,1,2,3 과 같이 정수의 형태로 제공될 때 사용한다.

4.2.4 Optimizer

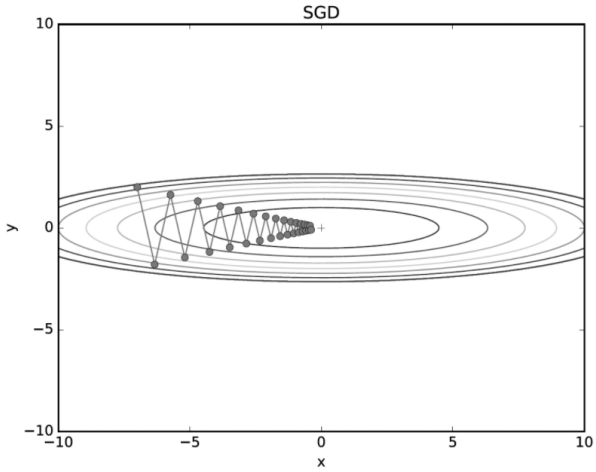
4.2.4.1 SGD(Stochastic Gradient Descent)

신경망의 가중치를 조정하는 과정에서 주로 사용하는 최적화 방법 중 하나이다. 네트워크에서 내놓는 결과값과 실제값 사이의 차이를 정의하는 손실함수의 값을 최소화하기 위해 기울기를 이용한다.

SGD 에서 최적화하는 과정은 다음과 같다. 우선 손실함수에서 시작점을 선택한다. 그리고 그 점에서의 기울기를 구한 뒤, 기울기 값에 학습률을 곱한 값을 뺀다. 이 과정을 반복하여 최소값에 점점 접근한다.



$$w = w - \alpha \frac{\partial L}{\partial w}$$



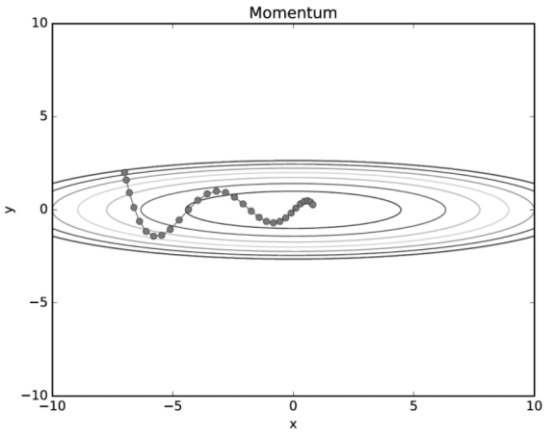
SGD 는 전체 데이터(Batch)가 아닌 일부 데이터의 모음(Mini-batch)를 사용한다. SGD 는 전체 데이터를 계산해야 하는 BGD(Batch Gradient Descent)에 비해 계산량이 적어 속도가 빠르기 때문에 같은 시간에 더 많은 step 을 나아갈 수 있다. 또한 데이터 세트에서 무작위로 균일하게 선택한 하나의 샘플을 이용하여 예측 경사를 계산하므로 local minima 에 빠질 확률이 적다. 하지만 일부의 데이터를 이용하기 때문에 다소 부정확할 수 있다는 특징이 있다.

4.2.4.2 Momentum

모멘텀은 운동량을 뜻하는 단어로, 아래 식처럼 계산된다.

$$v = \alpha v - \forall \frac{\partial L}{\partial w}$$

$$w = w + v$$



위 그림에서 볼 수 있듯이, 모멘텀의 갱신 경로는 공이 그릇 바닥을 구르듯이 움직인다. SGD 와 비교했을 때 지그재그의 빈도가 덜함을 알 수 잇다. x 축의 힘은 아주 작지만 방향은 변하지 않아 한 방향으로 일정하게 가속하기 때문이다. 거꾸로 y 축의 힘은 크지만 위 아래로 번갈아 받아서 상충하여 y 축 방향의 속도는 안정적이지 않다.

4.2.4.3 NAG

Nesterov Accelerated Gradient

기본적인 momentum 방식은 이동을 중지해야 하는 지점에 도달해도 momentum 에 의해 해당 지점을 지나칠 수 있다는 문제가 있다. Nesterov Accelerated Gradient 는 이러한 문제점을 해결하기 위해 제안되었다. NAG 에서는 momentum 계산 시에 momentum 에 의해 발생하는 변화를 미리 보고 momentum 을 결정한다.

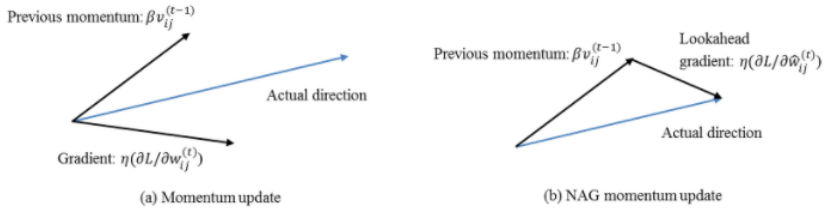
$$v = \beta v + \forall \frac{\partial L}{\partial W}$$

이때 w 는 다음과 같이 계산된다.

$$w = w - \beta v$$

그 다음 기본적인 momentum 방식과 같이 w 를 다음과 같이 update 한다.

$$w = w - v$$



기본적인 momentum 방식은 이전의 momentum 과 독립적으로 gradient 를 계산하여 momentum 과 gradient 를 더함으로써 update 방향을 결정한다. 반면 NAG 는 먼저 momentum 만큼 update 를 했다고 가정한 뒤, gradient 를 계산하고, 이를 기반으로 update 의 방향을 결정한다.

기본적인 momentum 방식은 현재의 가속도를 고려하지 않고 속도를 설정한다면, NAG 는 현재의 가속도를 어느 정도 고려하여 속도를 설정한다고 생각할 수 있다. 이러한 update 방식을 통해 NAG 는 momentum 을 이용한 빠른 이동이라는 장점을 유지하면서도 momentum 에 의해 과하게 이동한다는 단점을 해결한다.

4.2.4.4 AdaGrad

신경망 학습에서는 학습률이 매우 중요하다. 학습률이 너무 작으면 학습 시간이 길어지고 너무 크면 발산하게 되기 때문이다.

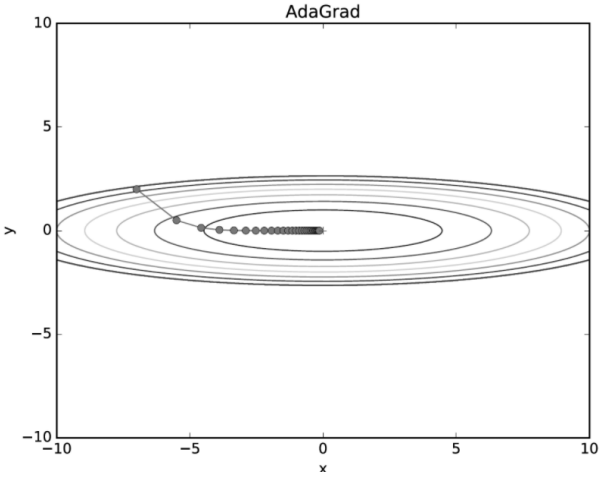
이 학습률을 정하는 효과적 기술로 학습을 진행하면서 학습률을 점차 줄여가는 방식이 있다. 매개변수 전체의 학습률 값을 일괄적으로 낮춰줌으로써 조절을 하는데 이러한 방식을 Adagrad 라고 한다.

$$h = h + \frac{\partial L}{\partial L}\frac{\partial L}{\partial L}$$

$$w = w - \forall \frac{1}{\sqrt{h}}\frac{\partial L}{\partial L}$$

Adagrad 는 과거의 기울기를 제곱하여 계속 더해간다. 그래서 학습을 진행할수록 갱신 강도가 약해진다.

그런데 실제로 무한히 Adagrad 를 이용하여 학습을 하다보면 갱신량이 0 이 되어 갱신이 멈추게 된다. 이러한 문제점을 개선한 기법이 바로 RMSProp 최적화 방법이다.



위 그림을 보면 최솟값을 향해 효율적으로 움직이는 것을 볼 수 있다. y 축 방향은 기울기가 커서 처음에는 크게 움직이지만, 그 큰 움직임에 비례해 갱신 정도도 큰 폭으로 작아지도록 조정된다.

그래서 y 축 방향으로 갱신 강도가 빠르게 약해지고, 앞의 SGD 나 momentum 보다 지그재그의 정도가 줄어든 것을 알 수 있다.

4.2.4.5 RMSProp

Root Mean Square Propagation

RMSProp 은 학습이 진행될수록 learning rate 가 극단적으로 감소하는 Adagrad 의 문제점을 해결하기 위해 제안되었다. Adagrad 는 변수가 얼마나 많이 변화했는지를 gradient 의 제곱의 합을 이용했지만, RMSProp 은 gradient 제곱의 합이 아니라 gradient 제곱의 지수 이동 평균으로 다음과 같이 계산한다.

$$g_t = \beta g_{t-1} - (1-\beta)\left(\frac{\partial L}{\partial W}\right)^2$$

그 다음, Adagrad 처럼 가중치를 update 한다.

$$w_{t+1} = w_t - \frac{\forall}{\sqrt{g_t + \varepsilon}}\frac{\partial L}{\partial W}$$

여기서 ∀은 학습률이고 ε는 매우 작은 수인 1e-8 로 설정한다.

g 는 이전의 변화량과 현재의 변화량의 어떠한 평균으로 정의되기 때문에 learning rate 가 급격하게 감소하는 현상을 방지할 수 있다.

4.2.4.6 Adam

Adam 은 Momentum 과 Adagrad 두 기법을 융합한 아이디어로, 현재 deep neural network 학습에 가장 광범위하게 사용되고 있는 최적화 알고리즘이다.

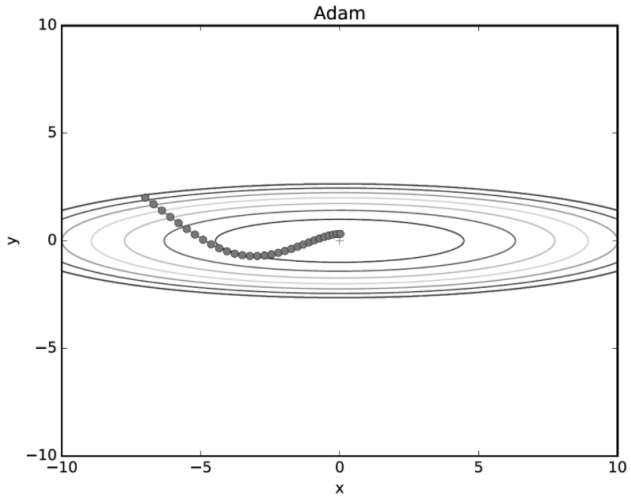
Adam 은 momentum 의 개념을 이용하지만, 가중치에 대한 momentum 과 learning rate 를 조절하는 계수가 지수 이동 평균으로 계산된다는 특징이 있다.

$$v_t = \beta_1 v_{t-1} + (1-\beta_1)\frac{\partial L}{\partial W}$$

$$g_t = \beta_2 g_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial W}\right)^2$$

최종적으로 Adam 방식을 통해 최적화를 하면 가중치는 아래와 같이 update 된다.

$$w_{t+1} = w_t - v_t\frac{\forall}{\sqrt{g_t + \epsilon}}$$



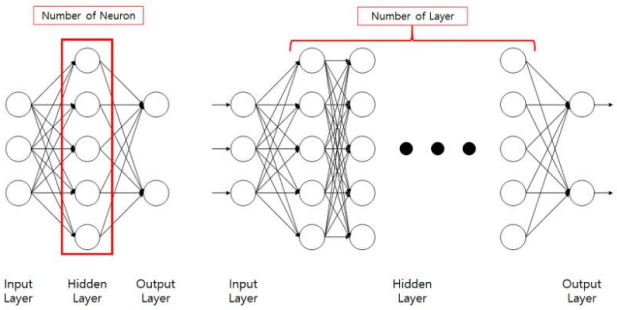
Adam 의 갱신 과정도 그릇 바닥을 구르듯 움직인다. Momentum 과 비슷해 보이지만 확실히 공의 좌우 흔들림이 적은 것을 알 수 있다.

4.2.5 정규화 파라미터

Overfitting 문제를 막기 위해 L1 또는 L2 정규화 방법을 사용한다.

4.2.6 은닉층 개수, 은닉층 노드 개수

은닉층은 입력층과 출력층 사이의 계층으로 은닉층의 개수와 각각 노드의 개수 또한 딥러닝 학습에 중요한 파라미터 중 하나이다. 뉴런의 개수와 레이어의 깊이는 Underfitting, Overfitting 에 영향을 준다.



4.2.7 Dropout

Dropout 은 overfitting 을 피하기 위한 정규화 기법이다. 학습 데이터에 과적합하게 학습된 모델을 실제로 사용할 수 있도록 일반화 능력을 높이기 위한 기법으로 은닉층에서 뉴런들 중 일부를 일정 비율로 배제시키고 학습을 한다. Dropout 비율이 너무 크면 Underfitting 이 발생하고, 너무 작으면 Overfitting 이 발생할 수 있기 때문에 적절한 값의 dropout 비율값을 설정해야 한다.

5. 기말프로젝트를 진행하기 위한 본인의 관심있는 주제 및 관련 기술 조사

체내에 약물을 주입할 때 기존의 방법은 주로 주사를 이용하여 투입한다. 하지만 이러한 방법은 혈류로 인해 필요한 부위에 집중하여 투입하는 것이 어렵다는 문제점이 있다. 이러한 문제점을 해결하기 위해 작은 물체(Object, 수 마이크로미터 크기)에 약물을 넣고 이 작은 물체를 체내에 넣어 필요한 부위 가까이 가서 터트리는 방법이 연구 중이다. 이렇게 Object 를 이용한다면 위의 사례 말고도 다른 분야에서 큰 도움이 될 것이라고 생각된다.

이때, 이 Object 의 움직임을 벡터 시퀀스 관점에서 분석해보고자 한다. Object 의 움직임을 관찰하는 요소로는 Cycle 횟수, Semi-Cycle 횟수, 속도, 가속도 등이 있다. 기존의 데이터를 observe 하여 학습을 하고, 이 object 의 경로와 위에서 언급한 요소들을 예측해보고자 한다.

논문 **Transformer Networks for Trajectory Forecasting** 에 의하면 경로 예측에 주로 쓰이는 LSTM 기반의 모델보다 Positional Encoding 방식을 이용하는 Encoder+Decoder 구조의 Transformer 기반의 모델이 훨씬 성능이 좋다고 한다. 이 논문을 기반으로 이 Object 의 움직임을 예측하고자 한다.

6.출처

1 장 출처

https://tensorflow.blog/%EC%BC%80%EB%9D%BC%EC%8A%A4-%EB%94%A5%EB%9F%AC%EB%8B%9D/1-3-%EC%99%9C-%EB%94%A5%EB%9F%AC%EB%8B%9D%EC%9D%BC%EA%B9%8C-%EC%99%9C-%EC%A7%80%EA%B8%88%EC%9D%BC%EA%B9%8C/

https://gomguard.tistory.com/184

https://ko.wikipedia.org/wiki/%EB%B9%85_%EB%8D%B0%EC%9D%B4%ED%84%B0

http://www.aitimes.com/news/articleView.html?idxno=133691

https://www.nvidea.com/

https://www.statista.com/statistics/871513/worldwide-data-created/

2 장 출처

https://colorofjuly.tistory.com/52

https://m.blog.naver.com/pasudo123/221069955611

https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=beyondlegend&logNo=221373971859

https://wikidocs.net/37406

https://box-world.tistory.com/8

https://ko.wikipedia.org/wiki/%ED%96%89%EB%A0%AC

3장 출처

https://roseline.oopy.io/dev/why-should-i-use-non-linear-function

https://ganghee-lee.tistory.com/32

http://www.incodom.kr/%ED%99%9C%EC%84%B1%ED%99%94%ED%95%A8%EC%88%98

https://360digitmg.com/activation-functions-neural-networks#ramp

https://yeomko.tistory.com/39

4 장 출처

http://www.gisdeveloper.co.kr/?p=8521

https://curt-park.github.io/2018-09-19/loss-cross-entropy/

https://www.clickai.ai/resource/wiki/model_interpretation/rmse_kor

http://taewan.kim/post/norm/

https://en.wikipedia.org/wiki/Mean_squared_error

https://dyndy.tistory.com/179

https://gooopy.tistory.com/61

https://ikkison.tistory.com/92

http://blog.skby.net/%ED%95%98%EC%9D%B4%ED%8D%BC%ED%8C%8C%EB%9D%BC%EB%AF%B8%ED%84%B0-hyperparameter/

https://bskyvision.com/822

https://wiserloner.tistory.com/1032

https://untitledtblog.tistory.com/149

https://gooopy.tistory.com/

https://wikidocs.net/35476