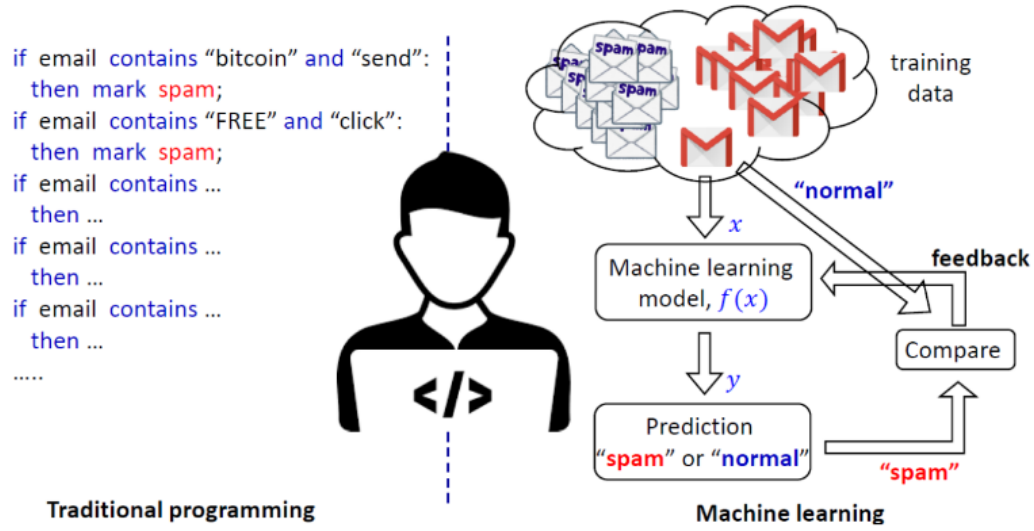
딥러닝팀 1 주차 클린업

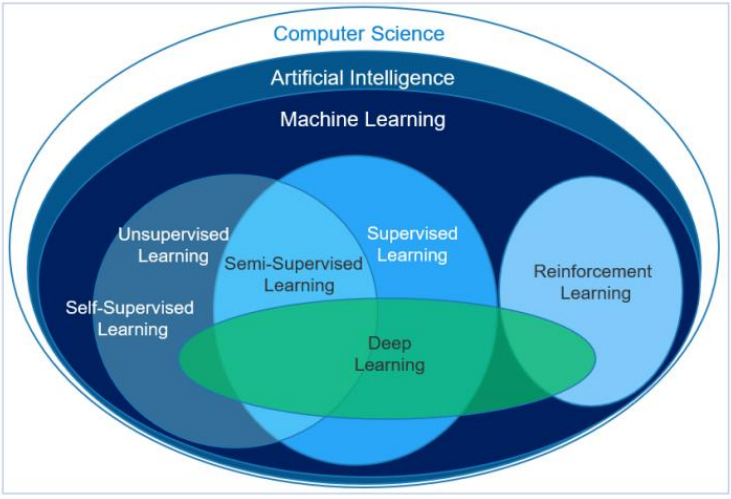
1. 머신러닝

머신러닝이란 데이터로부터 학습(Training or learning)을 통해 성능을 개선할 수 있는 컴퓨터 알고리즘의 한 분야입니다. 머신러닝은 '학습 데이터' 라고 불리는 샘플 데이터로부터 '모델'을 구축하고 학습하여 예측이나 결정을 내립니다.

간단하게 예시를 들어보겠습니다. 한번 스팸 메일을 분류하는 프로그램을 만든다고 생각해 봅시다. 전통적인 프로그래밍 방식은 왼쪽의 예시처럼 구분 규칙을 하나하나 다 적어주어야 합니다. 하지만, 머신러닝은 데이터에 의한 구분 규칙을 얻어내게 됩니다. 우리가 따로 구분 규칙을 적어주지 않아도, 데이터에 의해서 구분 규칙을 스스로 학습해 스팸 메일을 분류하게 됩니다.



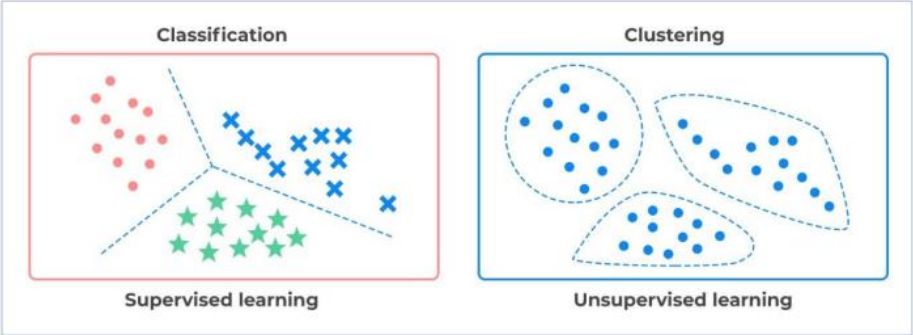
아래 벤다이어그램을 확인해보면 머신러닝에는 Supervised Learning, Unsupervised Learning, Reinforcement Learning등이 있습니다. 이 세가지에 대해 알아보고 마지막에 딥러닝이 무엇인지 알아보도록 하겠습니다.



1.1 지도학습과 비지도학습

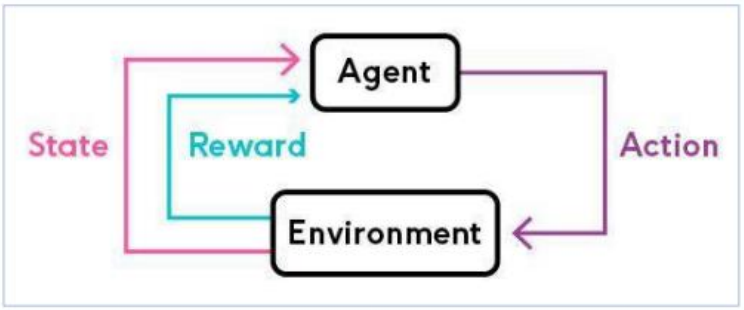
지도학습은 객체의 속성에 대한 입력과 출력이 데이터로써 주어졌을 때 그 입력과 출력 간의 함수관계를 유추하는 형태로 학습이 이루어집니다. 지도학습은 크게 분류 문제와 회귀 문제로 나눌 수 있습니다. 분류 문제는 예측 대상이 범주형 자료의 형태로 주어지는 예측과제를 의미합니다. 대표적인 분류 문제를 위한 머신러닝 알고리즘으로는 이진분류를 위한 로지스틱 회귀 모형, SVM, 그리고 다중분류를 위한 랜덤포레스트 분류 모델등이 있습니다. 반면 회귀 문제는 예측 대상이 연속형 자료로 주어지는 예측 과제를 의미합니다. 대표적인 회귀 문제 해결을 위한 모델로는 라쏘, 릿지 회귀 모형이 있습니다.

비지도학습은 이에 반해 출력값을 알려주지 않고 스스로 모델을 구축하여 학습하는 것을 의미합니다. 비지도학습은 입력만 있고 출력은 존재하지 않습니다. 따라서 규칙성을 스스로 찾아내는 것이 학습의 주요 목표입니다. 이 비지도학습의 결과는 지도학습의 입력으로 사용이 가능합니다. 이런 비지도학습의 과업으로는 대표적으로 군집화, 이상 탐지, 차원축소등이 존재합니다. 이상 탐지는 딥러닝에서도 자주 다뤄지는 주제이기 때문에 기억해두시면 좋을 것 같습니다. 이런 대표적인 지도학습, 비지도학습 모델들을 통틀어서 '고전적 머신러닝'이라고 부르도록 하겠습니다.



1.2 강화학습

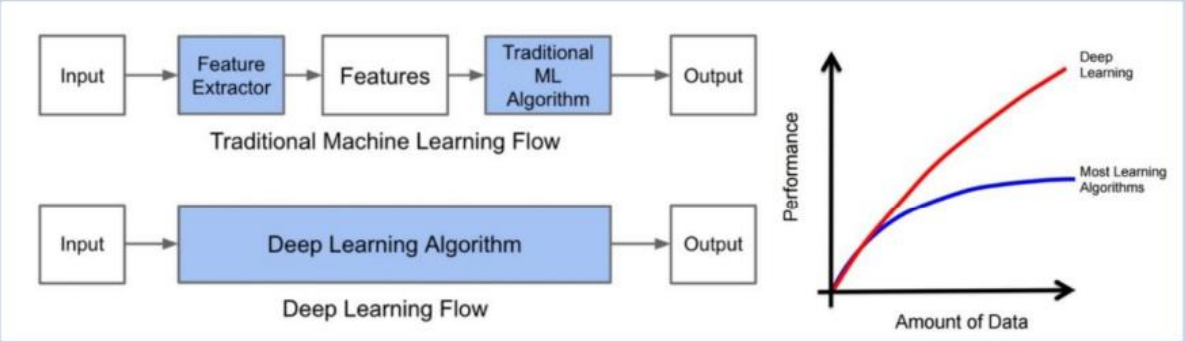
강화학습을 벤다이어그램에서 자세히 살펴보면 지도학습 및 비지도학습과 교집합이 존재하지 않는 것을 확인하실 수 있습니다. 강화학습이 '고전적 머신러닝'과 완전히 분리되어 있는 이유는 알고리즘의 작동 방식과 목표가 완전히 다르기 때문입니다. 강화학습이란 정답은 모르지만, 자신이 한 행동에 대한 "보상"을 알 수 있어서 그로부터 학습을 하는 모델입니다. 강화학습은 시행착오를 통해 보상하는 행동을 학습합니다. 최적의 값을 추구하기 위해 당근과 채찍을 사용한다고 생각하시면 됩니다.



1.3 딥러닝

딥러닝과 머신 러닝의 차이점은 feature engineering입니다. Feature engineering이란 모델의 성능을 높이기 위해서, 해석력을 높이기 위해서 데이터를 가공하는 것입니다. 데이터에서 새로운 feature를 만들어내거나, 유용한 feature를 사용하는 것 등입니다. (ex. 사람을 분류하기 위한 데이터에서 눈, 코, 입과 같은 사람을 구분할 수 있는 특징을 찾는 것) 머신 러닝에서는 이 feature engineering 과정에 따라 결과가 천차만별이지만, 딥러닝에서는 중요하지 않은 과정입니다. 그리고 feature engineering은 사람이 직접 해야 하기 때문에 많은 지식과 시간이 필요한 과정인 것에 반해, 딥러닝은 이 과정에서 사람이 개입하지 않아도 되기 때문에 강점을 가진다고 할 수 있습니다.

아래 그림은 딥러닝의 간단한 도식을 나타낸 것입니다.



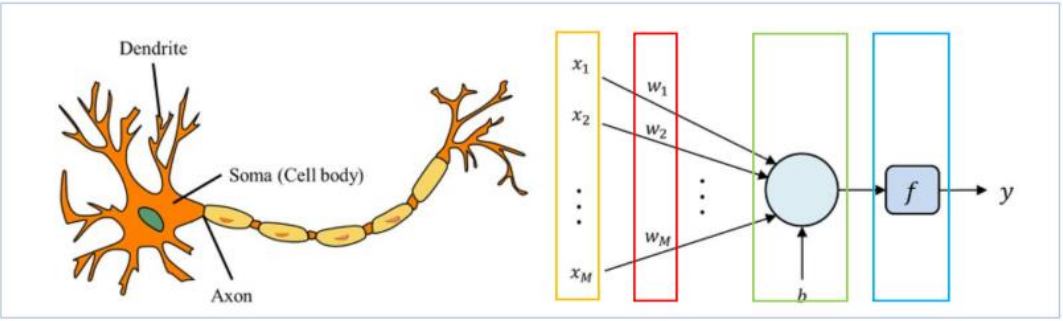
더 나아가, 딥러닝 알고리즘은 1 페이지 벤 다이어그램에서도 확인할 수 있듯이 지도학습, 비지도학습, 강화학습의 과제 모두에 적용될 수 있다는 장점 또한 존재합니다. 얼굴 인식, 질병 진단 등이 지도학습에서의 대표적인 활용 사례들이며, 이미지 생성, 음성 합성 등이 비지도학습의 대표적인 활용 사례라고 볼 수 있습니다. 또한, Deep Q-Network(DQN) 알고리즘과 알파고의 학습 원리인 REINFORCE 알고리즘은 강화학습에서의 대표적인 활용 사례입니다.

이제부터 딥러닝 알고리즘의 정의와 작동 방식에 대해 알아보도록 하겠습니다.

2. 퍼셉트론

2.1 단층 퍼셉트론

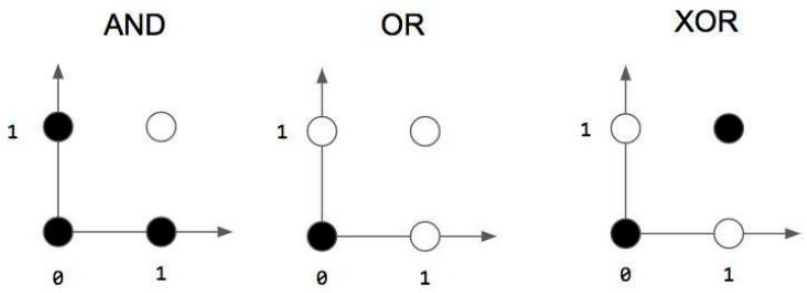
퍼셉트론(Perceptron)은 다수의 입력 데이터에 대해 하나의 출력을 반환하는 형태의 알고리즘으로, 신경 세포와 그 형태가 유사하다는 특징이 있습니다. 왼쪽은 신경 세포의 형태를 나타낸 그림이며, 오른쪽은 퍼셉트론을 도식화한 그림입니다. 보시는 바와 같이 퍼셉트론에는 노란색 구간에 해당하는 입력, 빨간색 구간에 해당하는 가중치(Weight), 하늘색 구간에 해당하는 활성화 함수(Activation Function)가 존재합니다. 연두색 구간의 경우 입력과 가중치를 곱하여 합하는 구간이기 때문에 별도의 명칭 대신 가중합(Weighted Sum)을 하는 구간으로 볼 수 있습니다.



퍼셉트론의 연산 과정을 한 마디로 표현하면, 입력과 가중치를 곱하여 모두 더한 후, 이 값을 활성화 함수에 통과시켜 임계값을 넘게되면 1, 넘지 못하게 되면 0을 출력하게 됩니다. 여기서 가중치와 임계값은 임의로 설정할 수 있는 매개변수(parameter) 이고 입력은 정해진 값입니다. 이를 식으로 표현하면 다음과 같습니다.

$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i + b \le \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i + b \ge \theta) \end{cases}$$

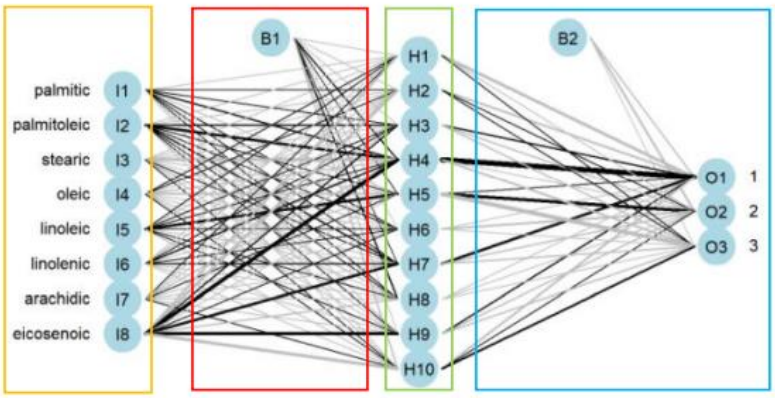
이 퍼셉트론을 이용하여 아래 사진의 문제를 해결해봅시다. 주어진 입력 값에 매개변수(가중치와 임계값)을 조절하여 검은 점(0)과 흰 점(1)에 맞는 값이 나오도록 하는 문제입니다. 선형구분선을 그어 검은 점과 흰 점을 구분하시면 됩니다.



퍼셉트론의 매개변수에 임의의 값을 넣게되면 $ax_1 + bx_2 > c$형태가 되는데 이는 직선의 방정식의 형태입니다. 즉 퍼셉트론은 선형적인 문제만을 해결할 수 있습니다. 그래서 AND와 OR 문제는 해결할 수 있지만 비선형적인 XOR 문제는 해결할 수 없다는 것이 퍼셉트론의 한계입니다.

2.1. 다층 퍼셉트론(multi-layer perceptron)

퍼셉트론 하나로는 XOR 문제를 해결할 수 없었지만, 여러 퍼셉트론을 연결하면 XOR 문제를 해결할 수 있습니다. 이렇게 여러 개의 퍼셉트론을 쌓아 올린 형태를 다층 퍼셉트론이라고 합니다. 다층 퍼셉트론을 이용하면 단층 퍼셉트론으로 해결할 수 없는 문제를 해결할 수 있는데 이것이 딥러닝의 기본적인 방식입니다. 다층 퍼셉트론의 동작 방식에 대해 좀 더 자세하게 알아본 이후 XOR 문제를 함께 해결해보겠습니다.

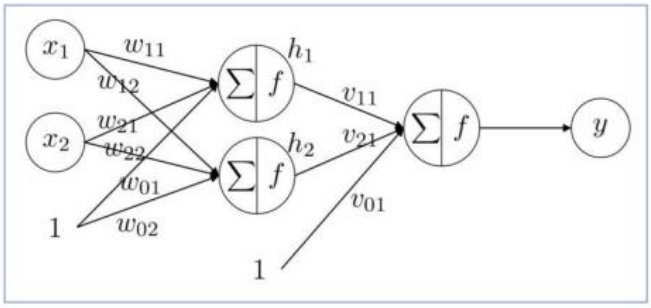


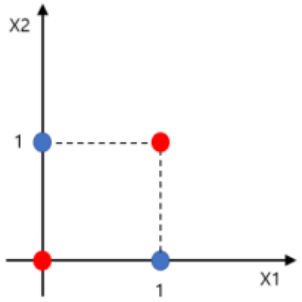
이 그림은 다층 퍼셉트론의 예시입니다. 여기서 입력과 출력에 해당하는 파란 원을 노드(Node)라고 부릅니다. 단층 퍼셉트론의 경우와 마찬가지로 노란색 상자는 입력, 빨간색 상자는 가중치를 의미합니다. 하지만, 연두색 상자부터 퍼셉트론과는 뭔가 다른 형태를 가지고 있습니다. 기존 퍼셉트론에서는 연두색 상자 부분에 y에 해당하는 출력이 나와야 하지만, 위 그림에서는 입력처럼 여러 개의 노드들이 있습니다. 그 이유는, 연두색 상자 속 H1부터 H10이 각각 서로 다른 퍼셉트론의 출력에 해당하기 때문입니다. 노란색 상자 속 원의 입력들은 연두색 상자의 원 하나하나와 빠짐없이 선으로 연결되어 있으며, 이 선들은 각각 10개의 퍼셉트론 속 8개의 가중치에 해당합니다.

한 가지 더 퍼셉트론과 차이가 나는 부분은 파란색 상자입니다. 위 문단에서 연두색 상자 속 H1 부터 H10이 10개의 퍼셉트론의 출력이라고 했는데, 파란색 상자를 보면 H1부터 H10에서 O1부터 O3에 대해 선이 연결되어 있음을 확인할 수 있습니다. 이는 H1부터 H10이 앞쪽 퍼셉트론의 출력이자 뒤쪽 퍼셉트론의 입력임을 의미합니다. 즉, I에서 H로 한 번, H에서 O로 한 번 총 두 번에 걸쳐 퍼셉트론을 쌓았음을 의미하는 것입니다.

여기서 노란색 상자를 입력층(Input Layer), 연두색 상자를 은닉층(Hidden Layer), 파란색 상자를 출력층(Output Layer)이라고 부릅니다. 즉, 한 층에는 여러 개의 퍼셉트론이 존재할 수 있으며, 은닉층의 노드들은 입력층에서 온 퍼셉트론들의 출력이자, 출력층으로 가는 퍼셉트론들의 입력인 것입니다.

이렇게 퍼셉트론을 중복으로 쌓아 다층 퍼셉트론으로 만들어준다면, 다층 퍼셉트론은 개별 퍼셉트론의 비선형성을 데이터에 여러 번 부여하여 선형 모델의 한계를 더욱 효과적으로 극복할 수 있게 됩니다. 이렇게 선형 모델의 한계를 극복하여 XOR 문제를 해결해 보겠습니다. 이해를 쉽게 하기 위해서 다시 한번 XOR 그림을 가져오도록 하겠습니다. 이번 XOR에서는 빨간색 원의 라벨을 0, 파란색 원의 라벨을 1로 놓겠습니다.





$w_{11} = 1.0,\ w_{21} = 1.0,$
$w_{01} = -1.5$

x_1	x_2	Σ	y_1
0	0	−1.5	0
0	1	−0.5	0
1	0	−0.5	0
1	1	0.5	1

$w_{12} = 1.0,\ w_{22} = 1.0,$
$w_{02} = -0.5$

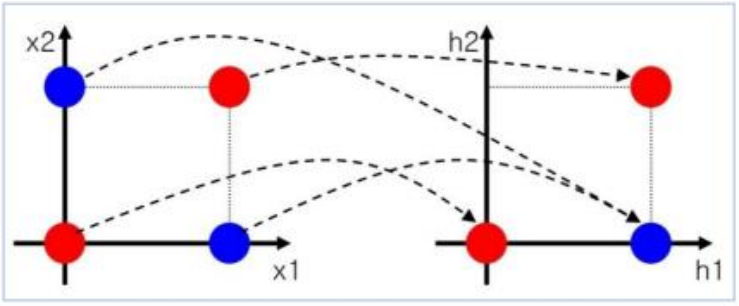
x_1	x_2	Σ	y_2
0	0	−0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	1.5	1

$v_{11} = -1.0,\ v_{21} = 1.0,$
$v_{01} = -0.5$

y_1	y_2	Σ	y
0	0	−0.5	0
0	1	0.5	1
0	1	0.5	1
1	1	−0.5	0

위의 그림과 같은 다층 퍼셉트론은 입력층과 은닉층의 노드가 2 개, 출력층의 노드가 1 개이며, 활성화 함수가 계단함수(Step Function)인 이진 분류 모델입니다. 첫번째 표는 주어진 입력층에서 은닉층으로 연결된 2 개의 퍼셉트론 중 첫번째 퍼셉트론의 가중치를 차례대로 1, 1, -1.5 로 놓은 결과, 두번째 표는 다른 하나의 퍼셉트론의 가중치를 1, 1, -0.5 로 놓은 결과입니다. 즉, 표에 y_1, y_2가 곧 다층 퍼셉트론 도식의 h_1, h_2에 해당하는 것입니다. 마지막으로 세번째 표는 은닉층에서 출력층으로 연결된 퍼셉트론의 가중치를 -1, 1, -0.5 로 놓았을 때의 결과입니다. 결과적으로 총 9개의 가중치의 값을 위 표에 주어진 것처럼 설정하면 파란색 원의 좌표를 넣었을 때 1을, 빨간색 원의 좌표를 넣었을 때 0을 정확하게 반환하고 있음을 확인할 수 있습니다.

그렇다면 이번엔 어떻게 다층 퍼셉트론이 XOR 문제를 해결할 수 있었는지 알아보겠습니다. 크게 보면 다층 퍼셉트론은 입력된 (x_1, x_2)이라는 좌표를 (h_1, h_2)로 바꿔준 후 y를 반환합니다. 즉, 원래 X1축과 X2축으로 이루어진 2 차원 평면 위에 있던 점들을 H1축과 H2축으로 이루어진 2 차원 평면 위의 점으로 옮긴 것입니다. 이를 이해하기 쉽게 그림으로 이해해보면 아래와 같습니다.



파란색 점에 해당하는 왼쪽 2 차원 평면에서의 (1, 0)과 (0, 1)은 오른쪽 2 차원 평면에서의 (0, 1)로 모두 옮겨졌습니다. 그리고 빨간색 점들 역시 H1 축과 H2 축에서 해당하는 위치로 옮겨졌습니다. 왼쪽 2 차원 평면에서는 선형적인 구분선을 그을 수 없었던 반면, 오른쪽의 2 차원 평면에서는 선형적인 구분선을 그을 수 있게 되었습니다!

물론 위와 같은 좌표축 변환이 오로지 비선형성의 추가를 통해서만 가능한 것은 아닙니다. 주어진 2 차원 평면 내의 좌표축 변환으로는 해결할 수 없었던 XOR 문제를 아예 새로운 좌표 평면으로 옮김으로써 XOR 문제를 해결할 수 있었다는 점에서 비선형성의 중첩이 가지는 위력을 확인할 수 있습니다.

이제 3장부터는 다층 퍼셉트론의 가중치는 어떻게 정해지는지, 어떤 활성화 함수가 있는지 알아보도록 하겠습니다.

3. 신경망 (Neural Network)

신경망, 혹은 인공신경망은 다층 퍼셉트론과 동일한 개념입니다. 그래서 퍼셉트론을 알아야 신경망 모델을 이해할 수 있는 것입니다. 두 모델의 가장 큰 차이점은 순전파(Feedforward Propagation)와 역전파(Back Propagation)에 있습니다. 이외에도 활성화 함수로 계단 함수를 쓸 때는 다층 퍼셉트론, 시그모이드 등을 쓸 때는 인공신경망이라는 구분, 그리고 층의 개수에 따른 구분 등 여러가지 의견들이 있지만, 순전파와 역전파가 가장 큰 차이점이라고 할 수 있습니다.

그렇다면, 순전파와 역전파가 무엇인지 알면 비로소 신경망 모델을 이해할 수 있을 것입니다. 순전파와 역전파라는 표현에서 짐작할 수 있듯이, 순전파는 순방향으로 이루어지는 전파, 역전파는 역방향으로 이루어지는 전파를 의미합니다. 그리고 여기서 전파는 특정 값이 모델을 통과하는 과정을 의미합니다. 따라서, 순전파는 입력이 순방향으로 모델을 통과하여 출력에 도달하는 과정을, 역전파는 순전파를 거쳐 모델이 예측한 결과를 실제 데이터와 비교한 후 그 오차를 모델의 가중치에 반영하기 위해 출력이 역방향으로 모델을 통과하여 입력에 도달하는 과정을 의미한다고 할 수 있습니다. 차례대로 순전파와 역전파를 자세히 알아보겠습니다.

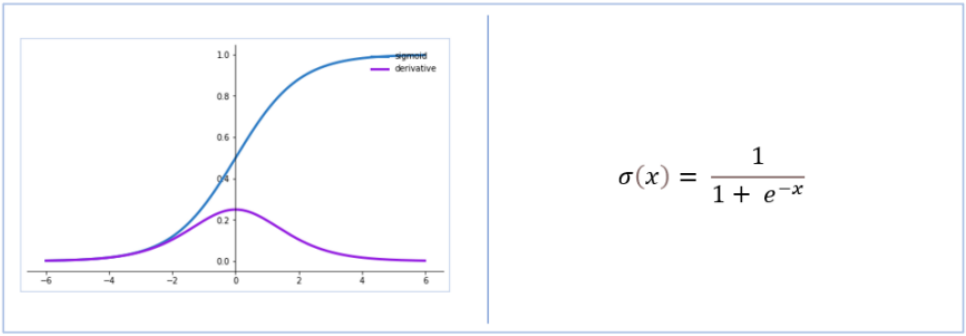
3.1. 순전파 (Feedforward Propagation)

순전파의 개념은 다층 퍼셉트론에서 살펴본 바와 같이 입력에 가중치가 곱해지고, 모두 더해져 활성화 함수를 거치는 과정 전반을 의미하는 것입니다. 다르게 말하면, 위에서 다룬 다층 퍼셉트론으로 XOR 문제를 해결했던 예시에서는 오로지 순전파 과정만을 거쳐 예측을 진행했다고 할 수 있습니다. 입력층부터 출력층까지 존재하는 모든 가중치의 값들을 임의로 설정한 후 입력이 들어갔을 때 모델을 통과한 후 옳은 값이 출력되는지 여부만을 확인했기 때문입니다. 따라서, 순전파를 보다 자세히 이해하기 위해 순전파 과정 중 우리가 알아보지 않고 넘어갔던 활성화 함수에 대해 보다 자세히 알아보도록 하겠습니다.

3.1.1. 활성화 함수 (Activation Function)

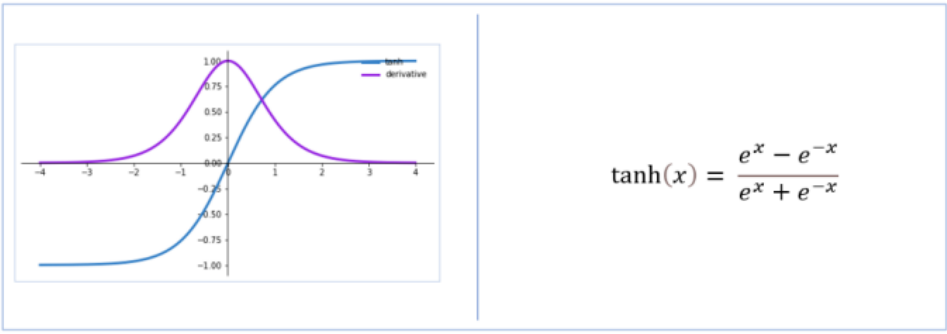
활성화 함수의 적절한 선택은 퍼셉트론의 장점을 극대화해줄 수 있는 2 가지 방법 중 하나에 해당했습니다. 입력과 가중치의 선형결합에 비선형성을 부여해주는 장본인이 바로 활성화 함수이기 때문입니다. 위에서는 계단함수만을 활용했지만, 지금부터 추가적으로 딥러닝 모델 전반에서 사용되는 몇 가지 활성화 함수에 대해 알아보도록 하겠습니다.

- 시그모이드 함수 (sigmoid)



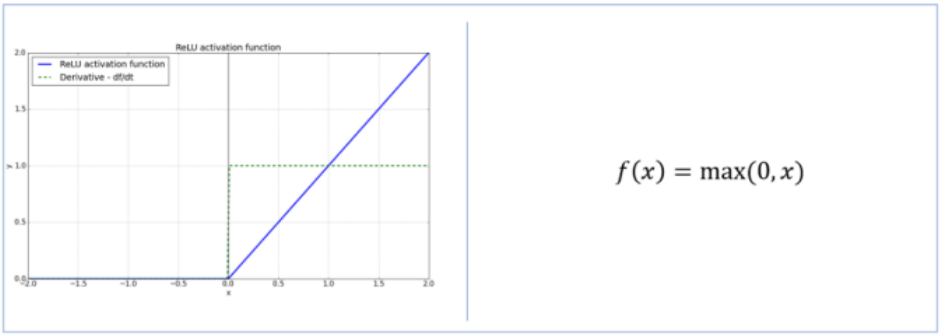
시드모이드 함수는 0에서 1 사이의 값을 가질 수 있는 함수입니다. 따라서, 이진 분류 모델에서 가장 기본이 되는 활성화 함수였습니다. 미분이 가능하기 때문에 역전파가 가능한데, 이 역전파에 대해서는 나중에 자세하게 다뤄볼 것입니다. 하지만 시그모이드 함수에는 여러 단점이 존재합니다. 먼저 출력이 0과 1 사이의 값이지만 그 사이의 값이 나오는 영역은 굉장히 작아 많은 값이 0 또는 1에 거의 수렴하게 됩니다. 또한 보라색 그래프를 보면 입력값이 조금이라도 커졌을 때 미분값이 0으로 수렴한다는 것을 확인할 수 있습니다. 중앙값이 0.5이기 때문에 최적화에 어려움이 있고, 분모에 지수함수가 존재하기 때문에 많은 연산량이 필요합니다.

- 하이퍼볼릭 탄젠트 함수 (tanh)



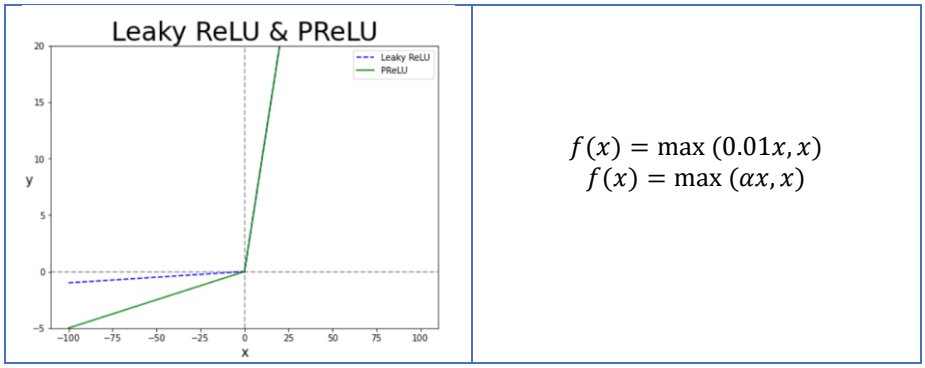
하이퍼볼릭 탄젠트 함수는 시그모이드 함수와 유사한 형태를 하고 있습니다. 다른 점이라면 중앙값이 0으로 음 수를 출력할 수 있다는 것입니다. 출력값의 범위가 더 크기 때문에 학습을 더 효율적으로 할 수 있습니다. 하지만 시그모이드 함수와 똑같이 입력값이 커지면 미분값이 0에 수렴한다는 것, 그리고 지수함수가 존재하여 많은 연산량이 필요하다는 단점이 있습니다.

- ReLU 함수



ReLU 함수는 딥러닝의 발전에 있어서 큰 문제였던 기울기 소실 문제를 해결한 활성화 함수입니다. 입력 값이 음수라면 0, 양수이면 자기 자신을 반환하는 함수입니다. 이유를 간단 하게 설명하면 출력 값의 범위가 아주 넓지만 특정 수로 수렴하지 않기 때문입니다. 또 이 ReLU 함수의 가장 큰 장점은 계산이 쉽다는 것인데, 이로 인해 모델의 학습 속도를 크게 줄일 수 있어 은닉층의 활성화 함수로 자주 사용됩니다. 하지만 ReLU 함수의 단점으로는 Knock out 문제가 발생할 수 있다는 것이 있습니다. 미분했을 때 0 or 1의 값을 가진다는 것은 어떤 레이어 하나에서 모든 노드의 미분 값이 0이 나온 다면 이후의 레이어에서 어떤 값이 나오건 학습이 이루어지지 않는다는 뜻입니다.

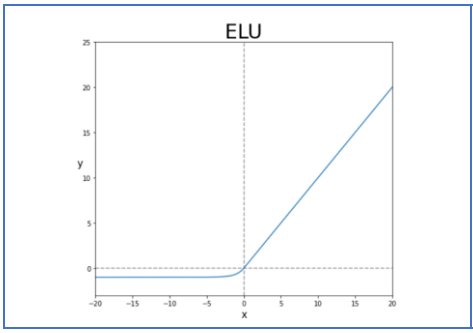
- Leaky ReLU 함수 & PReLU



ReLU 함수의 한계점인 음수값들이 다 0이 된다는 것을 해결하기 위해서 나온 활성화 함수가 바로 Leaky ReLU입니다. 기존 ReLU함수는 음수를 모두 0으로 해주었다면, Leaky ReLU는 음수를 0.01배 한다는 특징이 있습니다. 그러나, 음수에서 선형성이 생기게 되고, 그로 인해 복잡한 분류에서 사용할 수 없다는 한계점이 있습니다.

PReLU는 Leaky ReLU의 0.01이라는 고정된 값을 α로 하여, 하이퍼 파라미터로써 내가 원하는 값을 줄 수 있도록 만든 활성화 함수입니다. 하지만, 단점은 Leaky ReLU와 똑같습니다.

- ELU 함수

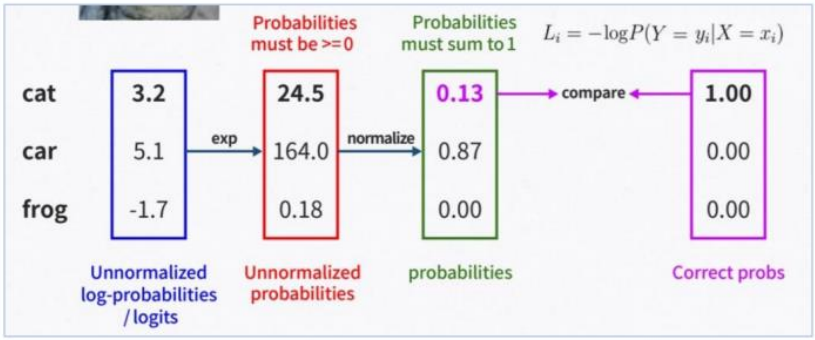


$$f(x) = \begin{cases} x & (x > 0) \\ \alpha(e^x - 1) & (x \leq 0) \end{cases}$$

ELU 함수는 ReLU 함수의 Knock out 문제를 해결하면서, 미분시 0에서 끊어지는 문제를 해결한 활성화 함수입니다. 0보다 작은 값에서 지수함수를 사용하여 그래프를 부드럽게 만들었습니다. 또한 Leaky ReLU와 PReLU와 다르게 음에서도 비선형적이기 때문에 복잡한 분류에서 사용할 수 있습니다. 하지만 ReLU에 비해 크게 성능이 증가한다는 이슈가 없고, 지수함수가 있기에 연산량이 늘어났다는 단점이 있습니다.

따라서 CNN 내용을 다룬 CS231N 수업에서는 먼저 ReLU를 사용하고, 그 이후 Leaky ReLU or ELU 순으로 사용할 것을 권장하고 있습니다. CNN 내용은 추후 2주차에 다룰 예정이지만, 관심이 있으신 분들은 CS231N 수업을 한번 보시는 것도 권장드립니다.

- 소프트맥스 함수 (Softmax Function)



소프트맥스 함수는 이전 활성화 함수들과 달리 출력층에서 활용되는 활성화 함수입니다. 그리고, 이진 분류 문제가 아닌 다중 분류 문제에 자주 활용된다는 특징을 가집니다. 소프트맥스 함수는 일련의 단계를 거쳐 출력층에서 반환되는 값을 [0, 1] 사이의 값으로 변환해줍니다. 위 사진의 예시를 보면서 알아보도록 하겠습니다. 가장 왼쪽 파란색 상자 안의 값은 신경망 모델의 출력층에서 반환된 값들입니다. 총 3 가지 라벨 중 고양이에 대해서는 3.2, 차는 5.1, 개구리는 -1.7 이라는 값을 반환했습니다. 하지만 우리가 학습에 사용하는 데이터는 가장 오른쪽 보라색 상자의 값들처럼 정답 라벨에는 1, 나머지 라벨에는 0 이 할당되어 있는 형태의 데이터입니다. 따라서, 소프트맥스 함수의 궁극적인 목표는 파란 상자의 출력층 값들을 보라색 상자의 라벨 데이터와 비교 가능한 형태로 바꿔주는 것입니다. 그 과정은 구체적으로 지수함수 통과, 정규화의 두 단계로 구성됩니다.

첫째로 지수함수 통과는 실수 전체 범위의 출력값을 양수 범위로 변환해주는 역할을 하며, 빨간색 상자 속 값들을 반환해 줍니다. 둘째로 정규화 단계는 양수 범위의 값들을 [0, 1] 이내의 값으로 변환해 주는 역할을 합니다. 따라서, 이 두 단계를 거치게 되면 각 라벨에 대한 값을 일종의 확률과 같이 이해할 수 있으며, 비로소 실제 데이터의 라벨과 비교가 가능하게 됩니다. 왜 데이터의 라벨과 비교하는 과정이 필요한지는 바로 다음 장에서 알아보도록 하겠습니다.

3.1.2. 손실함수 (Loss Function)

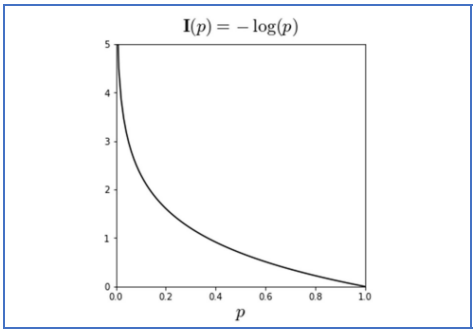
모델의 학습은 모델이 어떤 입력을 받았을 때 올바른 값을 출력하도록 함을 의미합니다. 이 과정에서 모델의 가중치가 바뀌게 됩니다. 이를 약간 다르게 표현하면 모델의 출력과 정답의 차이를 줄이도록 가중치를 바꾸는 과정을 모델의 학습이라고 할 수 있습니다. 학습을 위해서는 모델이 낸 출력과 정답의 차이를 계산할 수 있어야 합니다. 이 차이를 계산할 수 있도록 하는 것이 손실 함수입니다. 우리는 가장 흔히 사용되는 손실 함수인 평균 제곱 오차와 교차 엔트로피 오차에 대해서 알아보겠습니다.

- 평균 제곱 오차 (Mean Squared Error, MSE)

$$E = \frac{1}{2}\sum_{k=1}^{n}(\widehat{y_k} - y_k)^2$$

평균 제곱 오차는 회귀 문제에서 주로 쓰이는 손실 함수입니다. 앞에 1/2가 붙어있는데 역전파를 할 때 미분이 사용되는데 이때 값을 더 깔끔하게 해주기 위한 방법입니다.

- 교차 엔트로피 오차 (Cross Entropy Loss)



$$E = \sum_{k=1}^{n} y_k(-\log(\widehat{y_k}))$$

분류 문제에서 정답은 0 또는 1의 값을 가지게 됩니다. 반면 소프트맥스 함수를 통과한 모델의 예측 결과는 [0, 1] 사이의 실수로 반환됩니다. 오른쪽 교차 엔트로피 오차의 함수를 살펴보면 크게 y_k와 $-\log(\widehat{y_k})$의 곱으로 이루어져 있습니다. 그리고 $-\log(\widehat{y_k})$의 값은 왼쪽 그래프와 같이 0 에 가까울수록 더 높은 값을, 1 에 가까울수록 0 에 수렴하는 값을 가지게 됩니다. 즉, 정답과 모델의 예측 결과의 차이가 크게 날수록 오차가 커지도록 이루어져 있는 것입니다.

지금까지는 모델에 입력이 들어갔을 때 수많은 가중치들과 활성화 함수, 그리고 손실함수에 이르기까지의 순방향 과정을 살펴보았다면, 역전파에서는 모델의 가중치를 업데이트하는 것에 손실함수의 값을 어떻게 활용하는지 살펴보도록 하겠습니다.

3.2. 역전파 (Back Propagation)

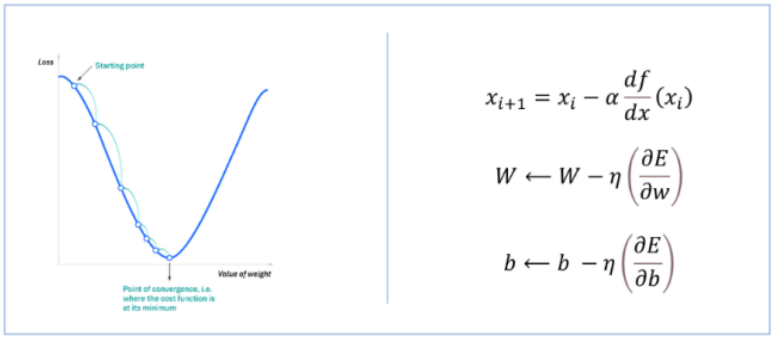
역전파란 모델이 학습을 해 나가는 과정입니다. 모델이 순전파를 진행하는 것을 모델이 추론해 나가는 과정이라고 했 는데, 우리는 이 과정을 거치고 나온 예측 값과 손실함수를 이용해 정답과 예측 값의 차이를 구할 수 있습니다. 모델의 학습 목적은 이 차이를 줄이는 것이고 이는 가중치와 편향을 업데이트하는 방식으로 이루어집니다. 업데이트 방식에는 여러가지가 있는데 기본적인 것들에 대해 다뤄보도록 하겠습니다.

3.2.1. Optimizer

딥러닝 모델에서 가중치와 편향을 업데이트 하는 과정은 가중치와 편향을 손실함수에 대해 최적화(optimization) 하는 것으로 볼 수 있습니다. 이러한 최적화를 수행하는 여러가지 방법들을 통틀어서 optimizer 라고 부릅니다. 이런 optimizer 에는 다양한 종류가 있는데, 그 중 가장 기본이 되는 경사 하강법(Gradient Descent)부터 확률적 경사 하강법(Stochastic Gradient Descent), Momentum 에 대해 알아보도록 하겠습니다.

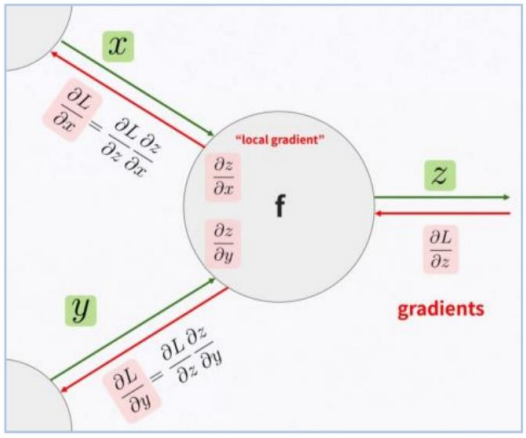
- 경사하강법

경사 하강법은 영어로 Gradient Descent 라고 합니다. 여기서 gradient 는 고차원 벡터의 편미분 값을 의미하며, descent 는 하강을 의미합니다. 따라서, 경사 하강법은 기본적으로 벡터의 편미분 값, 즉 기울기를 작게 만드는 방향으로 나아가는 형태의 최적화 방법을 의미합니다.

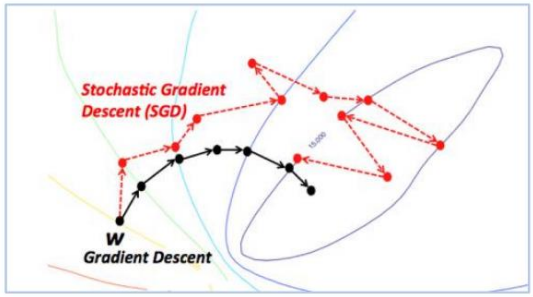


왼쪽의 그림에서 볼 수 있듯이, 시작점에서의 미분값은 절댓값이 큰 음수가 나올 것입니다. 경사 하강법의 기본적인 아이디어는 미분계수 부호의 반대 방향으로 반복하여 이동하며 최솟값을 찾는 것입니다. 시작점의 경우 최솟값으로부터 멀리 떨어져 있기 때문에 미분계수가 크지만, 이동을 거듭할수록 그 점에서의 미분계수는 점점 0 에 수렴해갈 것입니다. 이 때 미분계수에 비례하여 점이 다음 시점까지 움직일 거리를 정하기 위해 학습률(learning rate, α)이라는 개념을 사용합니다. 만약 업데이트 하는 가중치가 1 개라면 오른쪽의 수식으로 업데이트가 가능하겠지만, 딥러닝 모델의 수많은 가중치들을 층별로 업데이트하기 위해서는 가중치를 벡터의 형태로 놓고 편미분을 해야 합니다. 따라서, 각각의 가중치를 업데이트하는 과정은 오른쪽 두번째와 세번째 수식으로 나타낼 수 있습니다.

하지만 한 가지 고려해야 할 점은, 우리가 다루는 딥러닝 모델들은 여러 층으로 이루어져 있기 때문에 한 번의 미분으로는 모든 가중치에 대한 업데이트가 불가능합니다. 때문에, 미분의 연쇄법칙을 통해 손실함수의 값을 최종적으로 입력까지 전달하게 됩니다. 이는 그림을 통해서만 확인해보고 넘어가도록 하겠습니다.

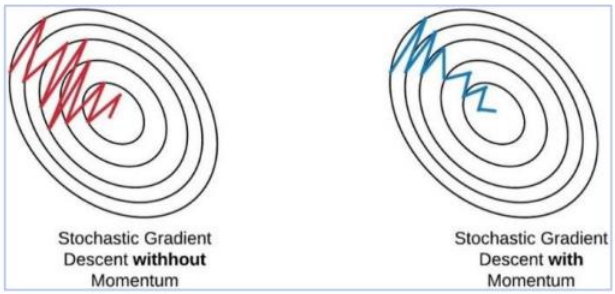


- 확률적 경사 하강법 (Stochastic Gradient Descent(SGD))



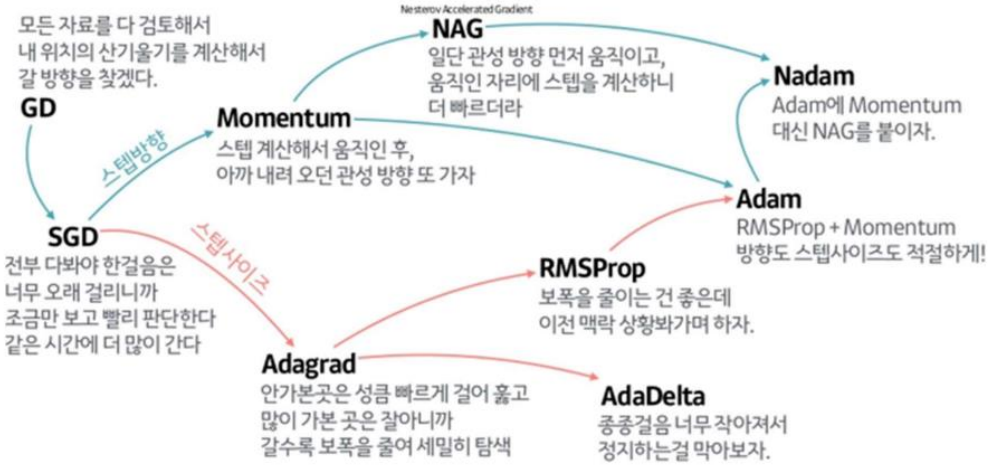
확률적 경사 하강법은 실제 딥러닝 모델 학습에서 이후 배울 Adam 이라는 optimizer 와 함께 가장 자주 사용되는 optimizer 입니다. 딥러닝 모델의 학습 데이터는 용량이 매우 크고 데이터의 수 또한 매우 많은 경우가 빈번하기 때문에, 모든 데이터를 활용하여 경사 하강법을 통한 학습을 진행하면 학습의 효율이 떨어질 수 있습니다. 따라서, SGD 는 일정 수의 데이터만을 사용하는 경사 하강법을 의미하는 optimizer 입니다. 그림에서도 볼 수 있듯이 모든 데이터를 사용할 때보다 변동이 심하다는 단점이 있지만, 효율성과 속도의 측면에서는 일반적인 경사 하강법보다 훨씬 뛰어납니다.

- Momentum



Momentum 은 단어 자체의 뜻처럼 관성을 의미합니다. 관성은 진행하던 방향으로 계속해서 진행하려는 성질을 의미합니다. 이를 최적화 문제에 적용하여 생각해보면, 기존 optimizer 에서는 미분값에 따라 한 단계 한 단계 최적점에 다가갔다면, 미분값이 클 경우 일종의 가속도를 부여하여 더 큰 보폭으로 움직이는 것을 의미합니다. 이는 딥러닝 모델을 학습할 때 자주 마주하게 되는 local minima 문제를 해결하는 데 큰 도움이 될 수 있습니다.

Optimizer 는 각각의 장점이 있지만 단점도 존재하기 때문에, 이를 해결하기 위해서 새로운 optimizer 들이 새로 나오고 있습니다. 아래 나오는 그림을 통해 여러 Optimizer 들을 확인할 수 있습니다. 여기 있는 모든 optimizer 를 기억할 필요는 없고, 딥러닝에서 가장 많이 쓰이는 Adam 에 대해서만 기억해주시면 좋을 것 같습니다.



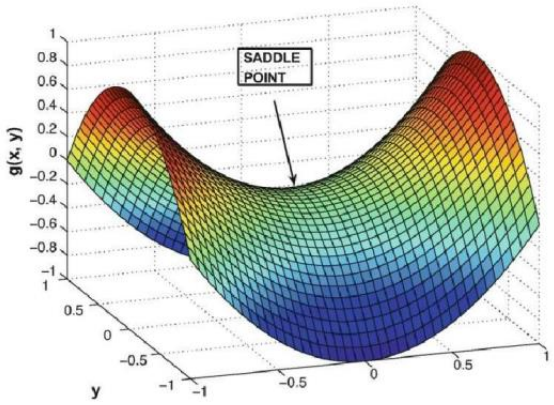
3.2.2. Local Minima 와 Saddle Point 문제

3.2.1 장에서 Optimizer 의 단점들이 존재하기 때문에 이를 해결하기 위해 새로운 optimizer 가 나왔다고 이야기했는데, 여기서는 그 대표적인 단점들에 대해서 이야기해보자 합니다. 그것들이 바로 Local Minima 와 Saddle Point 문제입니다. 각각에 대해서 한번 알아보도록 하겠습니다.

- Local Minima

Optimizer 를 활용한 최적화 방법들의 목표는 손실함수의 값이 최소가 되는 지점을 찾는 것이었습니다. 하지만, 함수에는 극소 지점이 여러 곳에 존재할 수 있습니다. 따라서 경사 하강법을 활용하여 최소 지점을 찾아가는 과정에서 학습률이 과하게 작은 경우에는 극소지점에서 머무르게 될 수도 있습니다. 이것이 바로 Local minima 입니다. 이전에 미분계수에 따라 나아가는 방향에 가속도를 적용하는 momentum 방법이 local minima 를 해결할 수 있는 방법이 될 수 있다고 말한 이유도 이것에 있습니다.

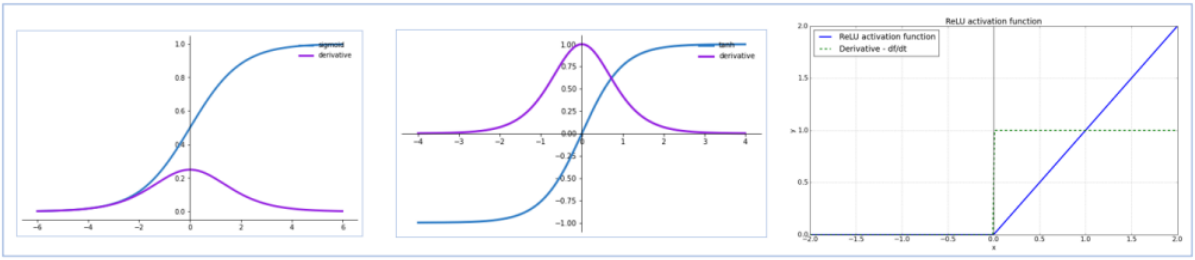
- Saddle point 문제



이 문제는 고차원 함수에서 나타날 수 있는 문제로, 특정 점이 어떤 방향에서는 최소값이지만 어떤 방향에서는 최대값이 되는 문제입니다. 이 경우 gradient 가 0 이 되기 때문에 더 좋은 값이 있을 수 있지만 학습이 멈추게 됩니다.

3.2.3. 기울기 소실 문제 (Gradient Vanishing Problem)

기울기 소실 문제는 역전파 과정에서 발생할 수 있는 문제입니다. 여기서 활성화 함수의 미분값들을 확인해보기 위해서 그래프를 다시 한번 가져와보겠습니다.



경사 하강법에서 다음 가중치는 이전 가중치에 대해 활성화 함수를 미분한 후 학습률을 곱하여 이동거리를 정하는 방법으로 업데이트가 이루어졌습니다. 즉, 미분계수가 현재의 점이 최적점으로부터 얼마나 떨어져 있는지를 알려주는 간접적인 지표가 되었다고 할 수 있습니다. 따라서, 경사 하강법이 정상적으로 작동하기 위해서는 미분계수가 최적점과 현위치의 차이를 적절하게 반영해줄 수 있어야 한다는 전제가 필요합니다.

하지만 시그모이드 함수의 미분값들을 자세히 살펴보면, 미분값이 아무리 커도 약 0.25 밖에 되지 않음을 확인할 수 있습니다. 또한, 0 을 중심으로 조금만 값이 벗어나면 미분값이 금방 0 에 수렴해버리는 모습을 보입니다. tanh 함수의 경우 미분값의 최고점이 시그모이드 함수에 비해서는 크지만, 0 으로부터 멀어졌을 때 미분값이 0 에 수렴한다는 점에서는 큰 차이를 보이지 않습니다. 역전파 시에는 미분의 연쇄 법칙에 의해 미분값, 즉 기울기가 출력층에서 여러 은닉층들을 거쳐 입력층까지 전파되어야 하는데, 시그모이드 함수를 한 번 통과할 때마다 [0, 0.25]의 값이 계속해서 곱해지기 때문에 기울기가 점점 0 에 수렴해갈 것입니다. 따라서 최적점과 현위치의 차이를 미분계수가 적절하게 반영해주지 못하게 되어 학습이 멈추게 됩니다.

마지막 그래프에 해당하는 ReLU 함수의 경우 미분값의 그래프가 0 또는 1 의 계단함수 형태로 나타나기 때문에 기울기 소실 문제를 상당 부분 극복했다고 할 수 있습니다. 이는 ReLU 함수의 단순한 형태 덕분에 연산 속도가 매우 빠르다는 점과 더불어 ReLU 함수가 최근 딥러닝 모델들에서 가장 대표적으로 활용되는 이유이기도 합니다.

4. 성능 향상 기법

4.1. 가중치 초기화 (Weight Initialization)

신경망이 파라미터 최적화를 하는 과정에서, 동일하게 경사 하강법을 통해 내려간다고 하더라도 가중치에 따라서 도달하는 최저점은 다르게 됩니다. 그렇기에 처음 위치를 잘 정하는 것도 매우 중요한 과정입니다. 이 때문에 학습 시작 시점의 가중치를 정해주는 것이 매우 중요합니다. 바로 그 과정을 가중치 초기화라고 합니다.

- Zero initialization

파라미터 값을 모두 0 으로 설정하고 시작하는 것입니다. 하지만, 파라미터의 값이 모두 같다면 역전파를 통해서 갱신하더라도 모두 같은 값으로 변하게 됩니다. 신경망 노드의 파라미터가 모두 동일하다면 여러 개의 노드를 사용하는 의미가 없어지게 됩니다. 따라서 이 방법은 딥러닝에서는 잘 사용하지 않습니다.

- Random initialization

파라미터 값을 정규분포를 따르는 랜덤으로 설정하고 시작하는 것입니다. 하지만 평균이 0, 표준편차가 1 인 정규분포를 따를 때는 sigmoid 가 활성화 함수로 layer 을 통과할수록 그 출력 값이 0, 1 에 치우치게 되고 따라서 그 지점에서의 gradient 도 0 에 가까워집니다. Gradient Vanishing 문제가 발생한다는 것입니다. 또한, 평균이 0, 표준편차가 0.01 인 정규분포를 따를 때는 그 출력값이 0.5 로 치우치게 되어 Zero initialization 과 같이 여러 개의 노드를 사용하는 의미가 없어지게 됩니다.

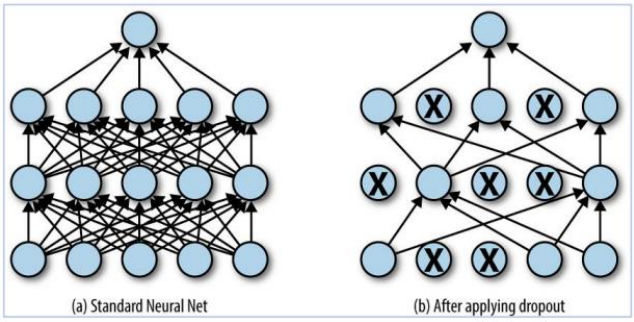
- Xavier initialization

사비에르 초기화는 고정된 표준편차를 사용하지 않고, 이전 은닉층의 노드의 개수가 n 개이고 현재 은닉층의 노드가 m 개일 때, $\frac{2}{\sqrt{n+m}}$ 을 표준편차로 하는 정규분포로 가중치를 초기화합니다. 층마다 노드 개수를 다르게 설정하더라도 이에 맞게 가중치가 초기화되기 때문에 고정된 표준편차를 사용하는 것보다 훨씬 더 강건(Robust)합니다. 사비에르 초기화는 활성화 함수가 sigmoid 함수일 때 많이 사용합니다.

- He initialization

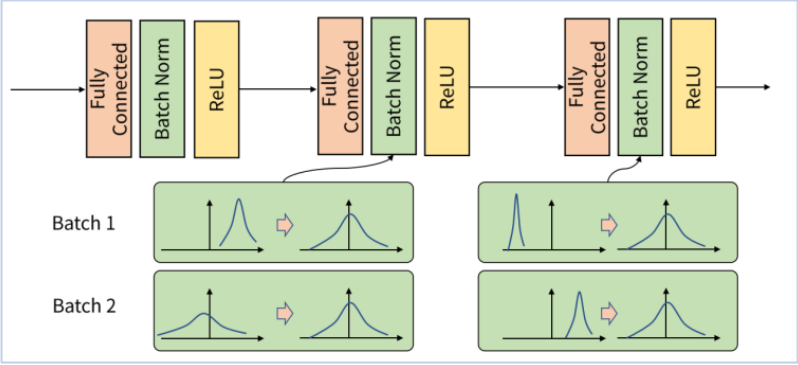
He 초기화는 $\sqrt{\frac{2}{n}}$를 표준편차로 하는 정규분포로 초기화하는 방법입니다. 활성화 함수가 sigmoid 함수일 때는 사비에르 초기화를 많이 사용했지만, 활성화 함수가 ReLU 함수일 때는 층이 깊어지게 될수록 활성값의 분포가 치우치게 됩니다. 이렇게 된다면 Gradient Vanishing 문제가 발생할 수 있습니다. 따라서 활성화 함수가 ReLU 함수일 때는 He 초기화를 많이 사용하게 됩니다.

4.2. Dropout

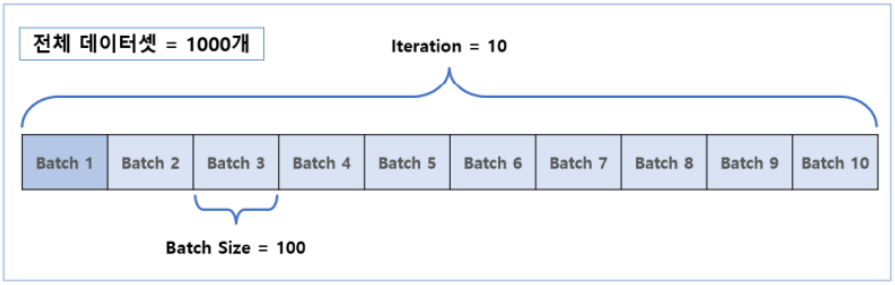


Dropout 은 batch 마다 학습할 때 일정한 비율의 노드(Node)를 버리고 학습하는 방법을 의미합니다. 이렇게 학습을 진행할 경우, 랜덤으로 일정한 비율의 노드들을 제외하고 학습하기 때문에 여러 모델을 앙상블(ensemble)한 것과 유사한 효과를 낼 수 있습니다. 따라서, 과적합을 방지할 수 있는 방법 중 하나로 이해할 수 있습니다.

4.3. Batch Normalization



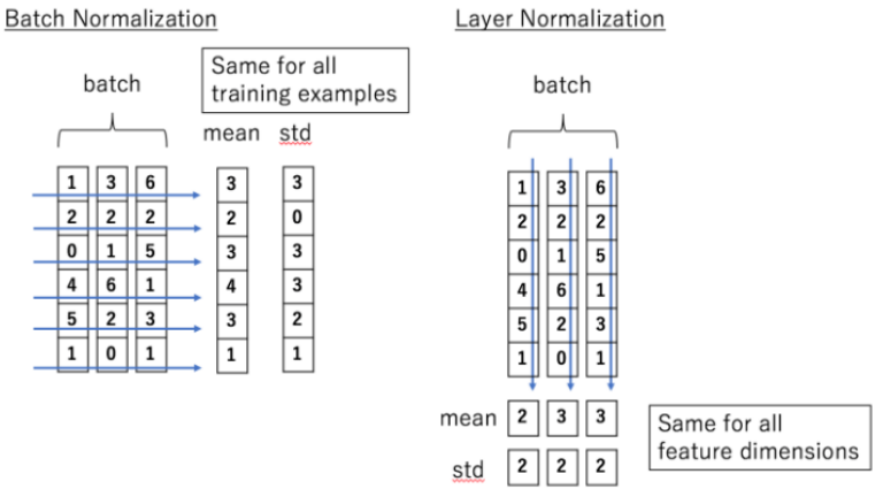
Batch Normalization은 batch마다 다른 분포로 인해 가중치 학습이 batch에 영향을 받는 것을 방지하기 위해 batch별 데이터를 평균과 분산으로 정규화 해주는 것을 의미합니다. 보통 각 layer의 활성화 함수 단계 직전에 넣어주게 됩니다. 여기서 batch는 매우 큰 입력 데이터셋을 나누는 단위를 말합니다.



위 그림과 같이 batch의 크기에 따라 Iteration의 크기가 결정됩니다. 즉, batch는 전체 데이터셋의 일부분이며, iteration 은 전체 데이터셋을 몇 개의 batch로 나누는지를 의미합니다. 위 예시를 살펴보면 전체 데이터셋이 1000개고, batch size를 100으로 설정했기 때문에 전체 iteration 수는 10이 되는 것을 확인할 수 있습니다.

학습 과정에서는 batch별 평균과 분산을 구하는 것이 쉽지만, 예측 과정에서는 입력 데이터에 대한 평균과 분산을 구하는 것이 쉽지 않습니다. 따라서, 예측 과정에는 학습 과정에서 얻은 batch 별 평균과 분산의 평균을 사용하게 됩니다. 표본 분포로 모집단의 분포를 추정할 때와 마찬가지로, 표본의 크기에 해당하는 batch의 크기가 너무 작으면 우리가 Batch Normalization에서 기대하는 효과를 얻기 힘들어집니다. 또한 추후에 배울 RNN처럼 sequential 데이터를 처리하는 경우에는 Batch Normalization을 적용시키기 어렵습니다.

따라서 RNN의 경우에는 Layer Normalization을 많이 사용합니다.



Layer Normalization은 feature 차원에서 정규화를 진행하는 것입니다. Sequence에 따라서 파라미터 크기가 달라질 수 있는 Batch Normalization과 다르게 Layer Normalization은 Sequence에 강건하기 때문에 RNN에 적용할 수 있는 방법입니다. 이후 Weight Normalization, Projected error function regularization 등 여러가지 방법이 나왔으니 추가적으로 공부를 더 하셔도 좋을 것 같습니다.

5. 마무리

이번 1 주차에는 '딥러닝'이라는 개념을 알기 위해서 필요한 부분을 공부해 보았습니다. 가장 먼저 지도학습, 비지도학습과 강화학습의 개념에 대해서 배우고 머신러닝과 딥러닝의 차이에 대해 알아보았습니다. 그리고 딥러닝의 가장 기본이 되는 퍼셉트론을 알아본 후, 퍼셉트론을 여러 개 연결한 다층 퍼셉트론에 대해 공부하며 어떻게 다층 퍼셉트론이 비선형 문제를 효과적으로 해결할 수 있는지 알게 되었습니다. 이후 본격적으로 인공신경망 모델의 학습 과정을 이해하기 위해 순전파와 역전파의 과정을 살펴보았습니다. 결과적으로, 순전파 과정에서는 활성화 함수와 손실함수가 주축이 되어 모델이 입력 데이터를 바탕으로 어떻게 예측 결과를 출력하는지 알게 되었으며, 역전파 과정에서는 어떻게 실제 정답과 예측 결과 간의 차이를 바탕으로 모델이 스스로 가중치를 업데이트하여 성능을 개선하는지 optimizer 개념을 통해 알게 되었습니다. 마지막으로 성능을 향상시키기 위한 여러가지 방법에 대해 알아보았습니다.

이렇게 해서 딥러닝 모델의 기본이 되는 층의 개념을 확실히 알아보았습니다. 오늘 공부한 것들을 종합하여 생각해보았을 때, 딥러닝 모델이라는 것은 결국 층을 여러 개 쌓아 스스로 학습할 수 있는 모델을 의미하는 것임을 알게 되었습니다. 역전파에서 배운 몇 가지 문제점들을 효과적으로 해결하는 동시에 어떤 종류의 층을 얼마나 더 깊게 쌓을 수 있는지에 따라 딥러닝 모델의 학습 성능이 더욱 개선될 수 있을 것입니다. 따라서 두 번째 스터디에서는 Pytorch 와 Tensorflow 를 이용한 간단한 딥러닝 모델을 구현해본 후, 2 주차에는 CNN, 3 주차에는 RNN 에 대해서 공부해보도록 하겠습니다.