PSAT 딥러닝팀

클린업 1주차 교안

딥러닝 1주차 클린업

1. 머신러닝과 딥러닝

- 1.1 지도학습과 비지도학습
- 1.2 강화학습
- 1.3 딥러닝
  - 1.3.1 머신러닝의 한계점
  - 1.3.2 딥러닝 모델의 한계점

2. 퍼셉트론

- 2.1 단층 퍼셉트론
- 2.2 다층 퍼셉트론
  - 2.2.1 노드

3. 신경망

- 3.1 순전파
  - 3.1.1 활성화함수
  - 3.1.2 손실함수
- 3.2 역전파
  - 3.2.1 Optimizer
  - 3.2.2 Local Minima와 Saddle Point 문제
  - 3.2.3 기울기 소실 문제

4. 성능향상기법

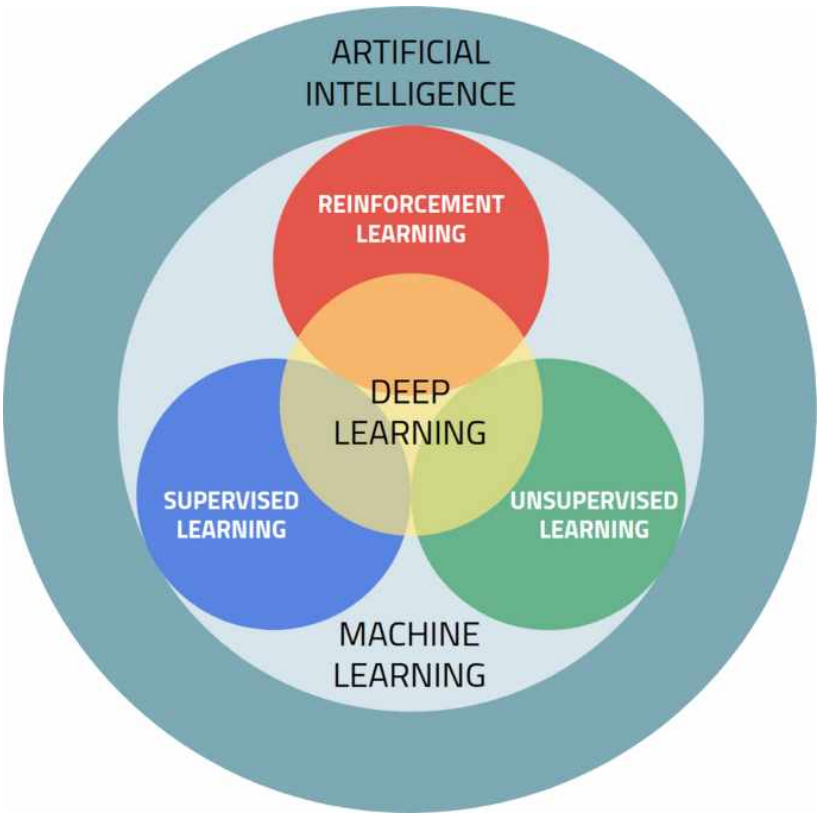
- 4.1 가중치 초기화
- 4.2 Drop Out
- 4.3 Batch Normalization

5. 마무리

1. 머신러닝과 딥러닝

인공지능은 문제해결, 패턴인식 등과 같이 인간의 지능과 연결된 인지문제를 연구하는 컴퓨터 공학 분야입니다. 1980년대, 인공지능을 구현하는 방법론으로 기호주의 학습이 주류를 차지했으나, 1990년대부터 현재까지 통계기반 머신러닝이 되었습니다. 기호주의 학습은 컴퓨터의 작동 방식으로 인공지능을 구현할 수 있다는 논리를 기반으로 하는데, 크게 두 가지 이유로 쇠퇴의 길을 걷게 되었습니다. 그 이유는 현실의 복잡한 지식을 논리 기호로 바꿀 수 없었고, 컴퓨터의 논리 규칙들이 현실 세계에 비해 충분히 복잡하지 않았기 때문입니다.

인간은 경험을 이용하여 자신이 겪는 문제를 해결하고, 새로운 행동 패턴을 학습합니다. 예를 들어, 잘 익은 수박을 고를 때, 두드렸을 때 소리가 맑은 수박을 잘 익은 수박이라고 판단할 수 있습니다. 머신러닝은 '인간이 경험을 활용하여 판단하고 예측하는 방식'을 이용하여 인공지능을 연구하는 분야입니다. 좀 더 구체적으로 머신러닝은 주어진 데이터를 이용하여 주어진 과업에 대해 성능을 향상시키는 알고리즘을 연구하는 분야로 이 알고리즘을 모델이라고 합니다.

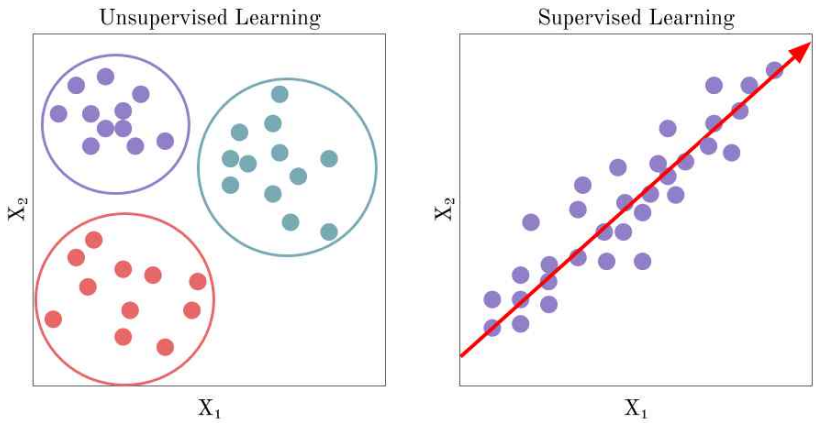


위의 다이어그램에서 알 수 있듯이 머신러닝의 하위 집합으로 크게 강화학습(Reinforcement Learning), 지도학습(Supervised Learning), 비지도 학습(Unsupervised Learning)과 딥러닝(Deep Learning)이 있습니다. 하나씩 간단히 살펴보며 딥러닝이 무엇인지 알아봅시다.

1.1 지도학습과 비지도학습

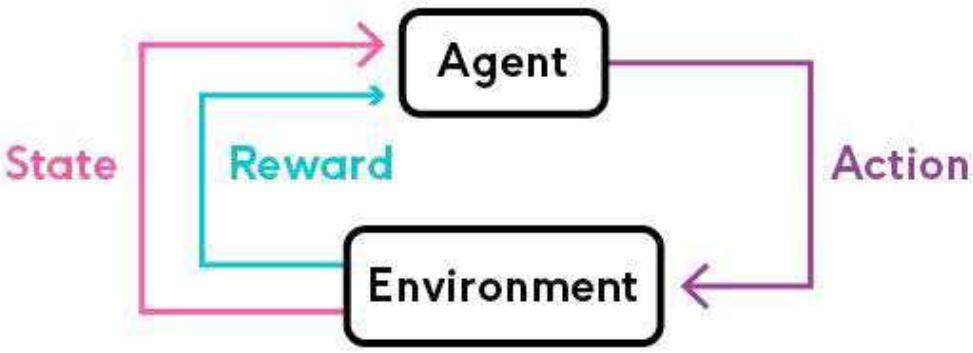
머신러닝의 과업(task)은 지도학습과 비지도학습으로 크게 분류될 수 있습니다. 지도학습은 객체의 속성에 대한 입력과 출력이 데이터로써 주어졌을 때 그 입력과 출력 간의 함수관계를 유추하는 형태로 학습이 이루어집니다. 그리고 지도학습은 크게 분류 문제와 회귀 문제로 나누어 이해할 수 있습니다. 분류 문제는 예측 대상이 양성/음성과 같은 범주형 자료의 형태로 주어지는 예측 과제를 의미합니다. 대표적인 분류 문제해결을 위한 머신러닝 알고리즘으로 이진 분류를 위한 로지스틱 회귀 모형, SVM, 그리고 다중 분류를 위한 랜덤포레스트 분류 모델 등이 있습니다. 반면 회귀 문제는 예측 대상이 키와 같은 연속형 자료로 주어지는 예측 과제를 의미합니다. 대표적인 회귀 문제해결을 위한 모델로는 라쏘, 릿지 회귀 모형이 있습니다.

반면 비지도학습은 객체의 속성에 대한 입력만이 데이터로 주어졌을 때 데이터를 설명하는 특성이나 패턴을 추출하는 형태로 학습이 이루어집니다. 비지도학습의 과업에는 대표적으로 군집화, 이상탐지, 연관분석, 차원축소 등이 존재하며, 이 중 특히 이상탐지는 딥러닝에서도 자주 다뤄지는 주제이므로 기억하고 넘어가면 좋을 것 같습니다. 지금까지 언급한 머신러닝의 대표적인 지도학습 및 비지도학습 모델들을 통틀어 지금부터 '고전적 머신러닝 모델'이라고 부르도록 하겠습니다.



1.2 강화학습

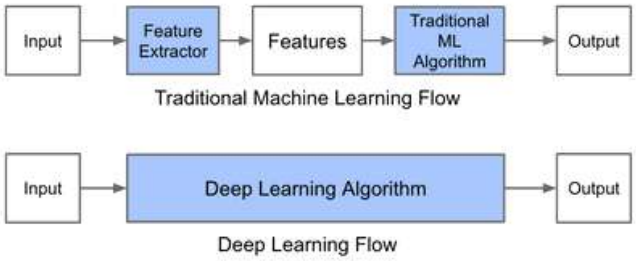
강화학습은 고전적 머신러닝 모델과 달리 입력과 출력이 따로 존재하지 않습니다. 대신, 특정 행동에 대한 보상이 주어져서, 이를 통해 학습을 진행합니다. 아래의 구조와 같이 Agent가 행동을 통해 환경으로부터 보상을 얻고, Agent가 처한 환경이 변화되게 됩니다. 강화학습은 Agent의 시행착오를 통해 결과적으로 보상을 최대화하는 방향으로 학습하게 됩니다.

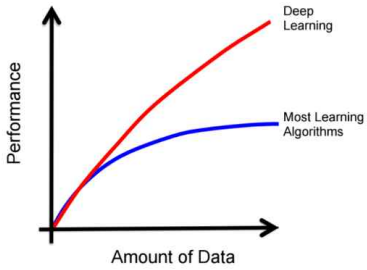


1.3 딥러닝

딥러닝과 고전적 머신러닝의 가장 큰 차이점은 특징 추출(Feature Extraction) 과정의 유무입니다. 고전적 머신러닝은 알고리즘의 구현 과정에서 사람의 수작업으로 입력 데이터에서 유의미한 특징을 추출하는 과정이 필요했습니다. 데이터 분석을 할 때 '전처리'라고 부르는 단계가 보통 이 과정에 해당합니다. 나중에 패키지 과제를 하면서 알게 되겠지만, 고전적 머신러닝 알고리즘의 경우 유의미한 변수를 추출하고 데이터의 형태를 가공해주는 과정이 모델의 예측 성능에 매우 큰 영향을 끼칩니다. 따라서, 고전적 머신러닝은 특징 추출 과정이 필수적입니다.

반면 딥러닝은 모델이 가공되지 않은 입력을 직접 처리하여 사람의 개입을 최소화해줄 수 있다는 장점이 있습니다. 이는 3 주 간의 스터디를 통해 단계적으로 알아가게 될 내용들이기 때문에, 아래의 도식으로 간단하게 이해하고 넘어가도록 하겠습니다.





고전적 머신러닝의 학습 과정을 살펴보면, 사용자가 데이터의 특징을 추출하고, 모델에 적절하게 가공된 데이터를 입력할 때 비로소 양질의 예측 결과를 얻을 수 있습니다. 하지만, 딥러닝 알고리즘은 특징 추출 과정이 생략되어 있음을 확인할 수 있습니다. 이를 End-to-End Learning 이라고도 합니다.

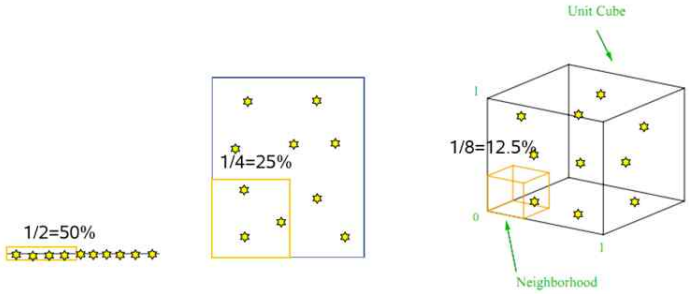
이처럼 사람의 개입을 최소화할 수 있다는 점은 빅데이터 분석에 대한 수요가 급증하는 시대적 흐름과 잘 맞았기 때문에 딥러닝 알고리즘에 대한 흥미와 수요가 급증하게 되었습니다. 그리고 실제로 딥러닝 알고리즘은 데이터의 크기와 형태가 커질수록 더 좋은 성능을 보였습니다.

더 나아가, 딥러닝 알고리즘은 앞선 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.

1.3.1 머신러닝의 한계점

앞서 설명한 특징 추출과정이 필요하다는 점이 머신러닝의 단점이라고할 수 있습니다. 또한, 음성이나 텍스트 같은 비정형 데이터에서 머신러닝 기법들이 잘 작동하지 않고, 고차원 공간에서 잘 작동하지 않습니다. 그 이유에 대해서 알아봅시다.

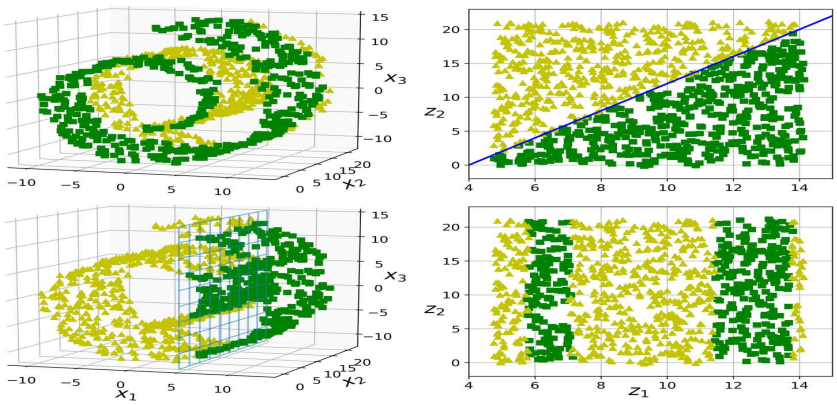
● 차원의 저주



차원의 저주란 간단히 말해 차원(입력 데이터의 특성 수)이 증가함에 따라 모델의 성능이 낮아지는 것을 의미합니다. 위의 그림은 각 차원마다 50%의 정보를 알고 있을 때, 차원이 증가할수록 (알고 있는 영역/전체영역)의 값이 줄어드는 것을 나타냅니다. 즉, 차원이 증가할수록 알고 있는 정보가 상대적으로 감소하여 모델의 성능이 저하되는 것입니다. 데이터의 희소성 이외에도 차원이 증가하면 데이터 간의 거리 측정이 의미를 상실합니다. 위의 그림에서 차원이 증가할수록 데이터 간의 거리는 평균적으로 점점 멀어지게 됩니다. 따라서, KNN과 같은 거리 기반 알고리즘은 작동하지 않을 것입니다.

차원의 저주를 해결하기 위해, 데이터 자체를 더 많이 확보하거나 변수 선택, 차원 축소 기법을 사용하여 문제가 되는 차원의 수를 줄일 수 있습니다.

● 다양체 가설



현실 세계의 의미 있는 데이터는 고차원에서 특정 부분에 몰려있을 가능성이 큽니다. 예를 들어, 알파벳을 무작위로 배열하여 생성한 문장이 의미 있을 확률은 0에 가깝습니다. 이를 고려하면 일상에서 접할 수 있는 영어 문장의 분포는 전체 영어 문장 공간에 비해 매우 작을 것입니다. 이처럼 의미 있는 데이터들이 다양체로 표현될 수 있고, 해당 다양체는 저차원으로 차원 축소를 통해 단순한 모양이 된다는 것이 다양체 가설입니다.

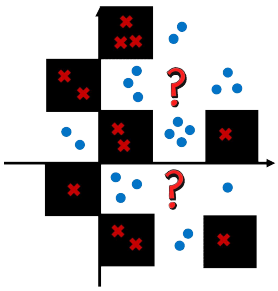
위 그림을 보면 롤케이크처럼 생긴 다양체를 펼치는 방식으로 차원 축소하여 단순한 모양으로 변환된 것을 알 수 있습니다. 그러나 아래쪽을 보면 같은 방법으로 다양체를 펼쳤을 때, 고차원일 때보다 복잡하게 변환된 것을 알 수 있습니다. 이 예시에서 3차원에서도 동일한 방식으로 다양체를 해석할 때 문제가 발생하는데, 고차원 데이터의 다양체를 기반으로 학습한다면 잘 작동하지 않을 것입니다.

● 국소일치성

기본적으로 통계학적인 모델이나 머신러닝에는 가정들이 들어갑니다. 예를 들어, 0주차에서 배운 선형 회귀 모델에서는 ϵ이 평균이 0인 정규분포를 따른다는 가정이 들어갑니다. 다른 예시로 머신러닝 기법 중 나이브베이즈는 조건부확률이 독립($p(x_i|y) \perp p(x_j|y),\ i \neq j$)이라는 가정이 들어갑니다. 조건부독립성과 같이 특정 모델에만 사용되는 가정이 아니라 거의 모든 머신러닝, 통계적 기법들이 암묵적으로 가지고 있는 가정이 있습니다. 바로 국소 불변성 사전분포(local constnacy prior)입니다.

$$f(x) \approx f(x+\epsilon)$$

이는 함수가 작은 영역 안에서는 아주 크게 변해서는 안 된다는 제약입니다. 이는 직관적으로 타당한 가정이지만 모든 데이터가 이 가정을 성립하는 것은 아닙니다.

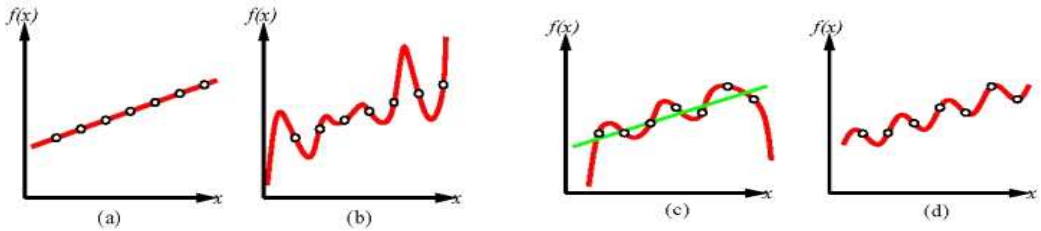


위 그림처럼 체스판 같은 확률공간에서 국소 불변성 가정을 이용한 머신러닝 기법들을 사용한다면 우리가 예측하려고 하는 공간을 흰색으로 예측하는 오류를 범할 수 있습니다. 물론 이런 경우에 데이터의 특성을 파악해 sin, cos과 같은 주기함수를 이용해 모델링을 한다면 이를 해결할 수 있을 것입니다. 하지만 고차원 데이터에서는 이러한 주기성 파악도 쉽지 않습니다. 따라서 머신러닝은 국소 불변성이 만족하지 않는 상황에서 효과적으로 작동할 수 없습니다.

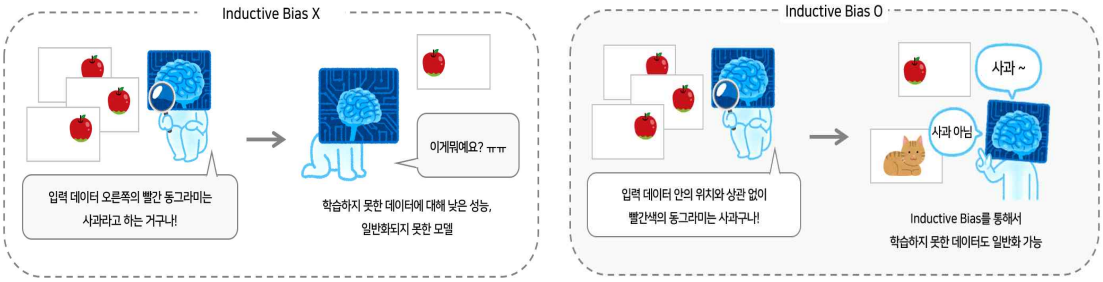
1.3.2 딥러닝 모델의 한계점

딥러닝 모델은 이미지와 텍스트 같은 비정형 데이터에서 좋은 성능을 보이지만, 정형데이터에서 대표적인 머신러닝 모델 분류인 트리 기반 모델보다 좋은 성능을 보이지 않습니다. 그 이유를 설명하기 전에 모델이 가지고 있는 귀납적 편향에 대해 알아봅시다.

● 귀납적 편향



귀납적 편향은 모델의 가정이나 구조에 존재하는 휴리스틱한 방법을 말합니다. 휴리스틱은 문제해결을 위해 경험적인 지식이나 규칙을 사용한다는 것입니다. 예를 들어 위 그림에서 주어진 데이터에 회귀곡선을 그린다고 할 때, 최고차를 1차로 제한하거나 차수를 좀 더 높여 복잡도를 주는 식으로 휴리스틱한 가정을 부여할 수 있습니다. 귀납적 편향은 학습 데이터 외의 데이터를 가지고도 모델의 성능을 일반화하기 위해 부여합니다. (b)의 그래프는 극단적인 예시로 일반화할 수 없는 귀납적 편향입니다. 추가로 앞서 소개한 국소불변성의 가정도 귀납적 편향이라고 할 수 있습니다.



위 그림의 왼쪽 모델은 학습 데이터의 오른쪽 빨간 동그라미는 사과라는 잘못된 귀납적 편향으로 인해, 새로운 데이터인 왼쪽에 존재하는 사과에 대하여 올바른 판단을 내릴 수 없습니다. 그러나, 우측의 2주차에서 배우게 될 CNN모델은 귀납적 편향인 이동 불변성을 모델 구조상에 가지고 있어, 사과의 위치와 관계없이 사과의 특징만을 학습할 수 있습니다. 따라서, 추후 위치가 다른 사과가 주어져도 좌측 모델과 다르게 사과라고 판단할 수 있습니다.

● 딥러닝 모델의 한계점

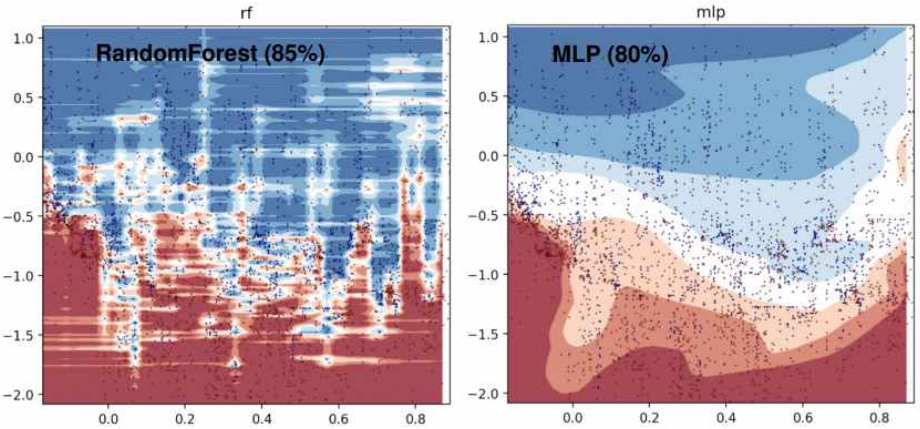
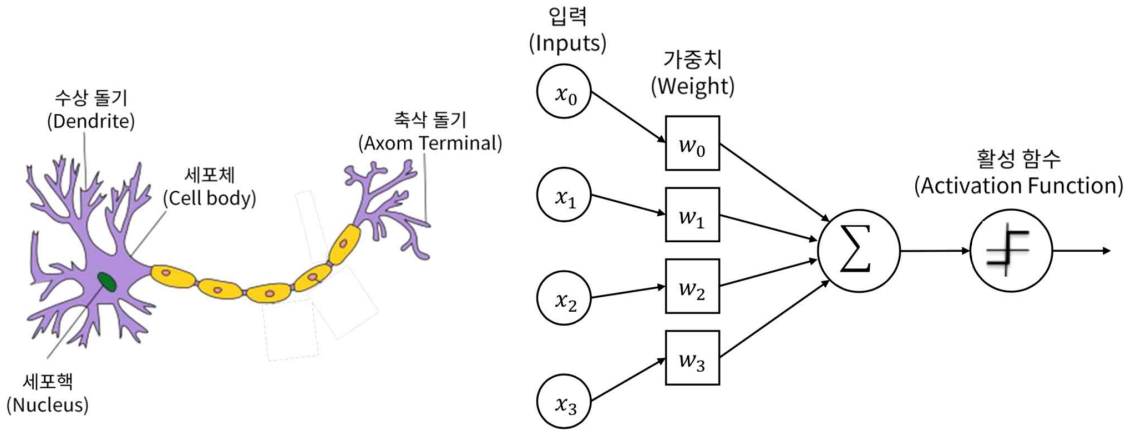


Figure 20: Decision boundaries of a default MLP and RandomForest for the 2 most important features of the *electricity* dataset

'Why do tree-based models still outperform deep learning on tabular data?'라는 논문에서는 위 그래프를 제시하며 기본적인 딥러닝 모델인 MLP(Multi Layer Perceptron)의 귀납적 편향이 정형데이터에 적절하지 않다고 주장합니다. 그래프에서 확인할 수 있듯이 MLP는 모델 구조상 부드러운 결정경계를 가정하게 되는데, 실제 정형데이터의 클래스 분포는 더 복잡한 것을 알 수 있습니다. 따라서 의사결정나무 기반의 Random Forest 모델이 더 좋은 성능을 보였습니다. End-to-End Learning의 장점과 성능을 정형데이터에 적용하기 위해, 정형데이터의 특징을 고려한 딥러닝 모델의 연구가 이루어지고 있으며, 그 예시로 TabNet이 있습니다. 이제부터 딥러닝 알고리즘의 정의와 작동 방식에 대해 알아보도록 하겠습니다.

2. 퍼셉트론

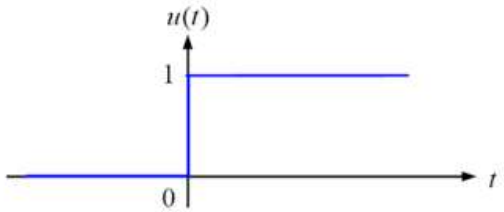
2.1 단층 퍼셉트론



딥러닝은 인간 신경의 기본 단위인 뉴런을 모방한 인공신경망 구조를 깊게 쌓아 인간의 지능을 구현하고자 합니다. 먼저, 인간의 뉴런은 신호를 입력받는 수상돌기와 다른 뉴런으로 신호를 전달하는 축삭돌기로 구성되어 있습니다. 다른 뉴런으로 신호를 보내기 위해서는 입력받은 신호를 처리한 전기적 값이 '활동전위'보다 높아야 합니다. 퍼셉트론은 인간의 뉴런을 수학적으로 모델링한 것이며, 뉴런과 마찬가지로 데이터의 입력을 받아 처리하여 값을 내보낼 수 있습니다. 3장에서 자세히 설명할 활성화 함수는 뉴런의 활동전위를 수학적으로 구현한 것입니다. 먼저, 퍼셉트론 내부에서 데이터 입력이 어떻게 처리되는지 알아봅시다.

$$z_i = w_0 x_{i0} + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + b$$

$$g_{\boldsymbol{w}}(\boldsymbol{x}_i) = \begin{cases} 0\,, (z_i \leq 0) \\ 1\,, (z_i > 0) \end{cases}$$



퍼셉트론의 계산은 가중치 $\boldsymbol{w}$에 데이터 입력 $\boldsymbol{x}_i$와 편향 b의 가중합을 계단함수에 통과하는 것으로 이루어집니다. 이 과정은 로지스틱 회귀에서 로지스틱함수를 계단함수로 변경한 것과 같습니다. 결국 퍼셉트론의 출력은 0아니면 1로 인간의 뉴런처럼 입력에 따라 활성화 여부를 출력하게 됩니다. 학습 파라미터인 $\boldsymbol{w}$는 다음 알고리즘에 따라 조정됩니다.

1. 전체 데이터의 i번째 데이터 $\boldsymbol{x}_i$를 입력으로 예측값 $g_{\boldsymbol{w}}(\boldsymbol{x}_i)$을 계산합니다,
2. 예측값과 실제값의 오차에 학습률 η와 입력데이터 x_{ij}를 곱한 값을 w_j에 더해 업데이트 합니다.
3. 1번과 2번의 과정을 모든 데이터에 대해 반복합니다.

해당 알고리즘을 수식으로 정리하면 다음과 같습니다. 입력 데이터를 오차에 곱하여 오차의 크기를 조정할 수 있습니다.

$$new \;\; w_j = w_j + \eta(y_i - g_{\boldsymbol{w}}(\boldsymbol{x}_i))x_{ij}$$

● 배타적 논리합 문제

퍼셉트론으로 논리곱(AND)와 논리합(OR)을 표현할 수 있게되어 인공지능 발전에 불을 지피게 됩니다. 그러나 1969년 MIT 인공지능 연구소의 마빈 민스키와 페퍼트 세이모어는 여기에 찬물을 끼얹었습니다. 그것은 바로 배타적 논리합(XOR)을 퍼셉트론이 표현할 수 없었기 때문입니다. 아래의 XOR 문제를 살펴보고 이를 다층 퍼셉트론이 이를 어떻게 해결했는지 알아보도록 하겠습니다.

오른쪽 그림에서 한 개의 직선만으로는 AND 문제처럼 XOR문제는 검은 점과 흰 점을 완전히 분리하는 영역을 만들어낼 수 없습니다. 여기서 한 개의 직선만을 긋는 이유는 퍼셉트론이 다음과 같이 직선의 방정식을 띄게 되기 때문입니다.

$$w_1x_1 + w_2x_2 - b = 0$$

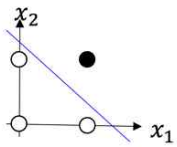
이 퍼셉트론의 한계로 인해 머신러닝은 첫 번째 암흑기를 맞이하게 됩니다. 그러나 아이러니하게도 이 문제의 해결책을 제시한 것도 민스키와 세이모어가 제시한 것입니다.

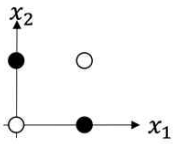
AND

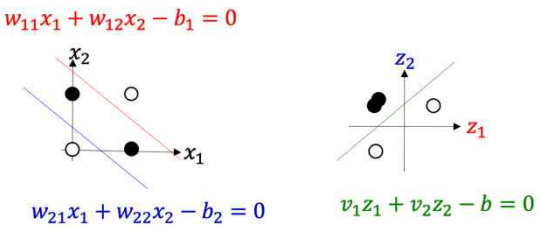
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

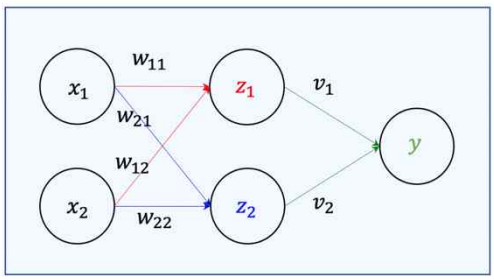
XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0









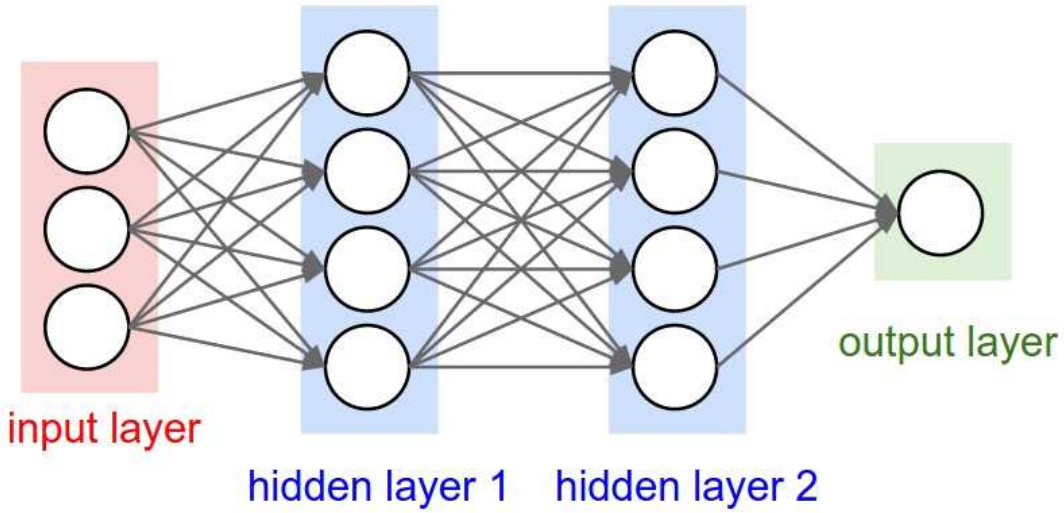
먼저 직선의 방정식 두 개를 사용하게 되면 흰 점과 검은 점의 공간을 완전히 분리할 수 있게 됩니다. 이 직선을 수식으로 표현하게 되면 다음과 같습니다.

$$s_1 = w_{11}x_1 + w_{12}x_2 - b_1 = 0$$
$$s_2 = w_{21}x_1 + w_{22}x_2 - b_2 = 0$$

이제 이 입력 (x_1, x_2)의 좌표변환 $z_1 = f(s_1)$과 $z_2 = f(s_2)$를 생각해보겠습니다. 그러면 위의 두 번째 그림과 같이 (z_1, z_2)에서는 흰 점과 검은 점을 가르는 하나의 직선을 구할 수 있습니다. 이를 딥러닝적으로 생각하면 입력 신호 (x_1, x_2)이 은닉층(hidden layer)를 지나 (z_1, z_2)로 변환되고 최종적으로 출력 신호 y로 변환됩니다. 즉 입력층과 출력층만으로 구성된 신경망에서는 표현할 수 없었던 배타적 논리합을 은닉층을 넣음으로써 표현할 수 있게 되었습니다. 이 입력층, 은닉층, 출력층에 대한 표현은 다층퍼셉트론에서 더욱 자세히 알아보겠습니다.

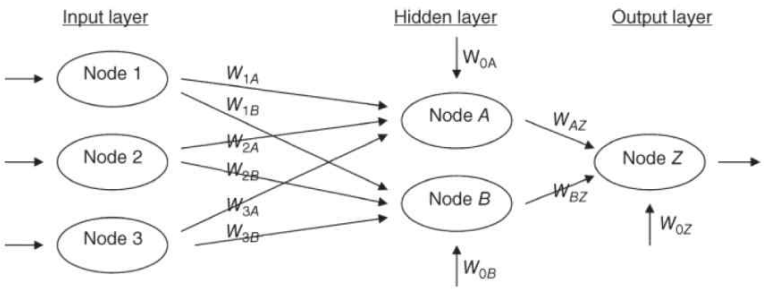
2.2 다층 퍼셉트론

XOR 문제에서 여러 개의 퍼셉트론을 쌓아 비선형성을 부여할 수 있었습니다. 이렇게 여러 개의 퍼셉트론을 쌓아 올린 형태를 다층 퍼셉트론 혹은 심층 순방향 신경망(deep feedforward [neural] network)이라고 합니다. XOR 문제에서는 각 퍼셉트론의 활성화 함수로 계단함수를 사용했지만, 실제 다층 퍼셉트론에서는 계단함수를 잘 사용하지 않습니다. 이제 다층 퍼셉트론의 구조를 살펴보겠습니다.



위의 그림은 다층 퍼셉트론의 예시로 1개의 입력층, 2개의 은닉층, 1개의 출력층으로 구성되어 있습니다. 첫 번째 은닉층은 4개의 노드를 가지고 있습니다. 노드에 대해서 알아봅시다.

2.2.1 노드



다층 퍼셉트론의 각 노드는 하나의 퍼셉트론이라고 이해할 수 있습니다. 위의 그림처럼 각 노드는 모든 데이터를 입력으로 출력값을 계산합니다. 다음 레이어의 노드는 이전 레이어의 노드의 모든 출력값을 입력으로 사용하여 다시 출력값을 계산합니다. 앞선 그림에서 각 노드는 모두 같은 역할을 수행합니다. 같은 레이어에 속한 노드의 경우 이전 레이어의 출력값을 입력으로 사용하기 때문에 하나의 레이어로 묶어놨다고 생각할 수 있습니다.

여기서, 각 레이어 노드들의 출력값을 잠재변수로 생각할 수 있습니다. 은닉층이 한 개라면 회귀모델과 비슷하게 각 노드의 가중치로 해당 노드가 어떤 의미를 지니는지 유추할 수 있습니다. 그러나, 층이 깊어지면 잠재변수로 다시 잠재변수를 설명해야 하기 때문에, 각 노드가 의미하는 바를 알 수 없습니다. 이러한 이유로 딥러닝 모델을 Black Box 모델이라고 부르며, 해석력을 부여하기 위한 '설명가능한 인공지능'(XAI, eXplainable AI) 연구가 이루어지고 있습니다.

3. 신경망

다층 퍼셉트론 모델에서 이 모델의 다른 이름으로 심층 순방향 신경망을 소개했습니다. 심층이라는 이름이 붙은 이유는 위에서 보았다시피 여러 층을 지나기 때문입니다. 그리고 순방향(feedforward)라는 명칭은 이 모델에서 정보가 앞쪽으로만 흐르기 때문입니다. 입력층에서부터 출력층까지 한 방향으로만 값이 처리되는 것을 위해서 확인할 수 있었습니다. 위에서 추상적으로 순전파(feedforward propagation)을 다뤘지만 이제는 구체적으로 어떻게 순전파 과정이 일어나는지 다뤄보겠습니다.

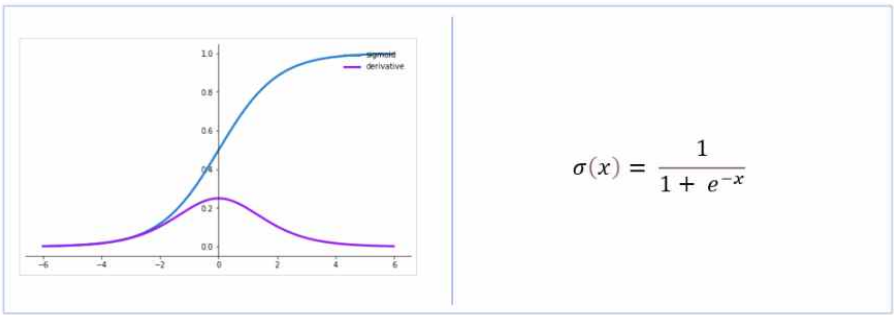
3.1 순전파

순전파란 입력 데이터를 기반으로 신경망을 따라 입력층부터 출력층까지 차례대로 변수들을 계산하고 추론한 결과를 의미합니다. 위에서 다룬 다층 퍼셉트론 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 자세히 이해하기 위해 순전파 과정 중 우리가 알아보지 않고 넘어갔던 활성화 함수에 대해 알아보도록 하겠습니다.

3.1.1. 활성화 함수 (Activation Function)

활성화 함수는 뉴런이 수상돌기에서 받은 전기 신호를 결과적으로 ON/OFF로 나타내는 것에 영감을 받은 함수입니다. 따라서 기초적인 활성화 함수인 계단함수는 오로지 0과 1만을 출력했습니다. 하지만 딥러닝이 발전하면서 활성화 함수는 비선형성을 부과하는 역할을 맡게 되었습니다. 이 덕분에 딥러닝이 비선형적인 함수를 근사할 수 있는 것입니다. 위에서는 계단함수와 로지스틱 함수를 활용했지만, 지금부터 추가적으로 딥러닝 모델 전반에서 사용되는 몇 가지 활성화 함수에 대해 알아보도록 하겠습니다.

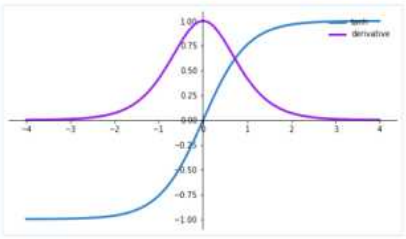
● 시그모이드 함수



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

시그모이드 함수는 로지스틱 함수의 다른 이름으로 이진 분류 모델에서 가장 기본이 되는 함수였습니다. 이는 계단함수와 유사하면서도 미분이 가능하기 때문에 역전파가 가능합니다. 이 역전파에 대해서는 나중에 자세히 다뤄볼 것입니다. 하지만 시그모이드 함수에는 여러 단점이 존재합니다. 먼저 출력이 0과 1 사이의 값이지만 그 사이의 값이 나오는 영역(약 -2에서 2)은 굉장히 작아 많은 값이 0 또는 1에 거의 수렴하게 됩니다. 즉, 이 함수는 가중합이 0에 가까울 때만 입력에 아주 민감하게 반응합니다. 또한 보라색 그래프를 보면 입력값이 조금이라도 커졌을 때 기울기 0으로 수렴한다는 것을 확인할 수 있습니다. 기울기가 0에 가깝다면 매개변수가 거의 학습하지 못합니다. 그래서 현재는 활성화 함수로 시그모이드 함수를 거의 사용하지 않습니다.

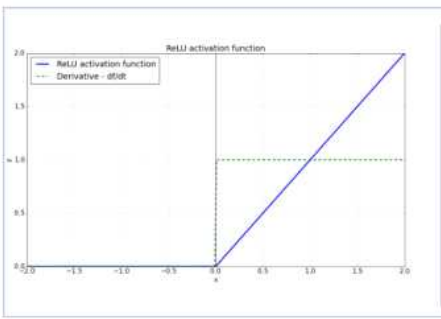
● 하이퍼볼릭 탄젠트 함수



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

하이퍼볼릭 탄젠트 함수는 시그모이드 함수와 밀접한 관계를 띄고 있습니다.[1] 다른 점이라면 $\tanh(0) = 0$으로 tanh는 0 근처에서 $y = x$와 비슷하다는 것입니다. 이러한 특성 덕분에 $\hat{y} = w^{T}\tanh(U^{T}\tanh(V^{T}x))$의 학습이 가중합들을 작게 유지할 수 있다면 $\hat{y} = w^{T}U^{T}V^{T}x$의 학습과 비슷해집니다. 이 덕분에 tanh 신경망의 훈련이 좀 더 쉬워집니다. 하지만 시그모이드 함수와 똑같이 입력값이 커지면 기울기가 0에 수렴해집니다. 또한 시그모이드 함수와 마찬가지로 지수함수를 계산해야 하기 때문에 연산이 느립니다.[2]

● ReLU 함수



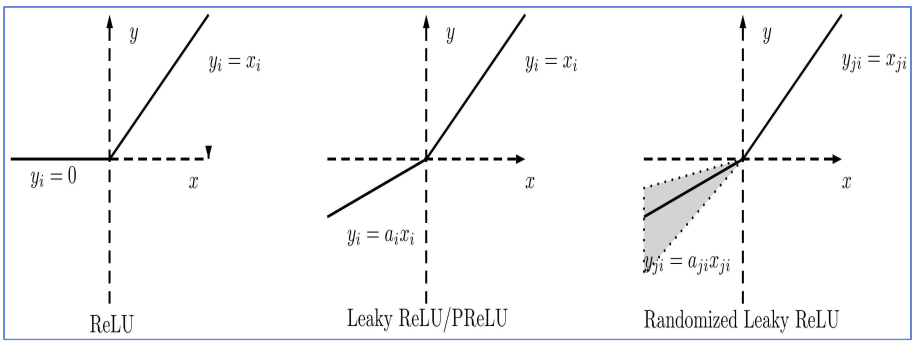
$$f(x) = \max(0, x)$$

ReLU 함수는 입력 값이 음수라면 0, 양수이면 자기 자신을 반환하는 함수입니다. 이런 ReLU 함수는 딥러닝의 발전에 있어서 큰 문제였던 기울기 소실 문제를 해결한 활성화 함수입니다. 이유를 설명하면 가중합이 0보다 큰 경우 기울기가 큰 값을 가지기 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 단점으로 가중합이 0과 같거나 작은 경우에서는 기울기 기반 학습을 할 수 없다는 것입니다.[3] 이런 단점을 극복하고 모든 점에서 반드시 기울기가 산출되도록 ReLU 함수를 개선한 활성화 함수가 존재합니다.

---

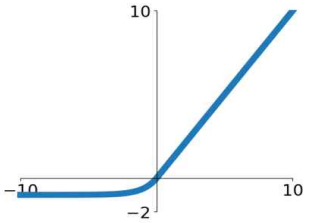
1) $\tanh(x) = 2\sigma(2x) - 1$
2) ReLU에 비해 약 3배정도 느리다고 합니다.
3) 이를 Knockout 문제라고 부릅니다.

● Leaky ReLU 함수 & PReLU(parametric ReLU)



ReLu 함수의 한계점이 음수값들이 다 0이 된다는 것을 해결하기 위해서 나온 활성화 함수가 바로 Leaky ReLU입니다. 기존 ReLu 함수는 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 매개변수로 하여, 스스로 적절한 값을 학습하도록 만든 활성화 함수입니다. 하지만, 단점은 Leaky ReLU와 똑같습니다.

● ELU 함수



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

(Default alpha=1)

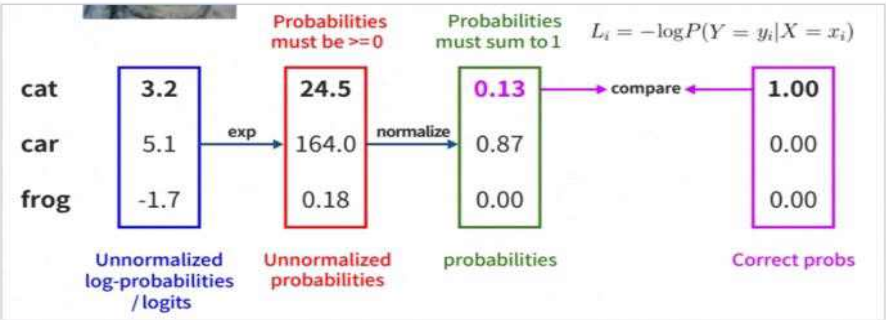
- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

- Computation requires exp()

ELU 함수는 ReLU 함수의 Knockout 문제를 해결하면서, 미분 시 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



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 출력층으로 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 3가지 라벨 중 고양에 대해서는 3.2, 차는 5,1, 개구리는 -1.7이라는 값을 반환했습니다. 하지만 우리가 학습에 사용하는 데이터는 가장 오른쪽 보라색 상자의 값들처럼 정답 라벨에는 1, 나머지 라벨에는 0이 할당되어 있는 형태의 데이터입니다. 따라서, 소프트맥스 함수의 궁극적인 목표는 파란 상자의 출력층 값들을 보라색 상자의 라벨 데이터와 비교 가능한 행태로 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 반환해줍니다. 둘째로 정규화 단계는 양수 범위의 값들을 [0,1] 이내의 값으로 변환해주는 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

손실함수(Loss Function)의 개념은 사실 딥러닝을 공부하지 않았더라도 회귀분석, 데이터 마이닝, 머신러닝 등에 대한 전공 수업을 수강한 경험이 있다면 익숙할 거라고 생각합니다. 딥러닝에서의 손실함수는 역전파를 위해 필수적인 단계입니다. 위에서도 간단히 언급했듯 역전파는 모델의 예측 결과를 실제 정답과 비교하여 얻어진 오차를 가중치에 반영하기 위한 과정입니다. 여기서 '모델의 예측 결과를 실제 정답과 비교'하는 역할을 담당하는 것이 손실함수입니다. 손실함수로는 분류 문제에 활용되는 교차 엔트로피 오차(Cross Entropy Loss), 회귀 문제에 활용되는 평균 제곱 오차(Mean Squared Error)를 알아보겠습니다.

● 평균 제곱 오차(Mean Squared Error, MSE)

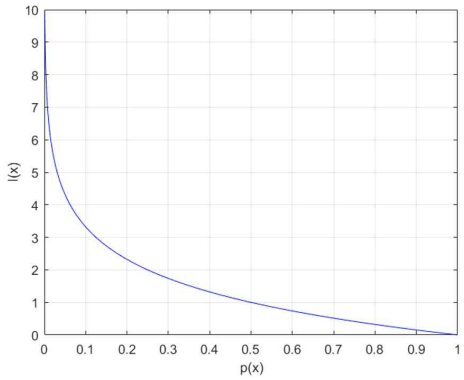
$$L = \frac{1}{2}\sum_{k=1}^{n}(\hat{y_k} - y_k)^2$$

평균 제곱 오차는 회귀 문제에서 주로 쓰이는 손실함수입니다. 이름 그대로 오차 제곱의 평균값을 손실함수 값으로 반환하기 때문에 제곱의 특성상 정답과 예측 결과의 차이가 클수록 오차의 크기가 더 많이 커지는 성질을 가집니다. 앞에 1/2가 붙어있는데 역전파를 할 때 미분이 사용되는데 이때 값을 더 깔끔하게 해주기 위한 방법입니다.

● 이진 교차 엔트로피 오차(Binary Cross Entropy Loss)

엔트로피란 사건이 가진 정보량을 평균화한 척도로 분류 문제에서 사용합니다. 어떤 사건 y가 발생하는 확률이 $p(y)$라고 했을 때, 사건 y의 정보량의 정의는 다음과 같습니다. 정보량은 놀람의 척도로 사건이 작은 확률을 가질수록 큰 정보량을 지니는 것을 아래 그래프를 통해 확인할 수 있습니다.

$$I(y) = \log(\frac{1}{p(y)}) \\ = -\log(p(y))$$



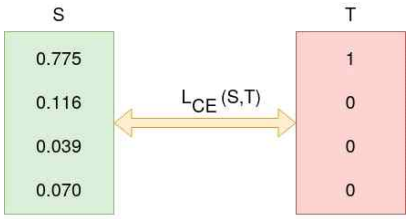
이진 분류 모델의 출력값은 0과 1사이의 값으로 예측값이 1인 확률이라고 생각할 수 있습니다. 이진 교차 엔트로피는 다음과 같이 정의됩니다. 실제값이 0인 경우 두 번째 항에서 정보량이 계산됩니다. 이때, 모델의 예측값이 1에 가까울수록 로그 안의 값이 작아져 큰 정보량을 갖습니다. 실제값이 0인데 예측값을 1이라고 한 것이므로 오차가 커지는 것이라고 이해할 수 있습니다. 반대의 경우도 마찬가지입니다. 로지스틱 회귀의 우도함수와 동일합니다.

$$BCE(\hat{y}) = ylog\frac{1}{\hat{y}} + (1-y)\log(\frac{1}{1-\hat{y}}) = -(ylog\hat{y} + (1-y)\log(1-\hat{y}))$$

● 교차 엔트로피 오차(Cross Entropy Loss)

다중 분류의 경우 손실함수로 교차 엔트로피를 사용합니다. 이 경우 일반화된 엔트로피의 정의로 오차를 계산하며, 사건의 정보량의 평균입니다. C는 다중 분류 대상의 범주 개수입니다.

$$Loss = -\sum_{c=1}^{C} y_c \log \hat{y_c} = E(\log \frac{1}{\hat{y}})$$



다중 분류 문제에서 출력층의 활성화함수는 요소의 합이 1인 벡터를 출력하는 소프트맥스입니다. 실제값 y도 C차원의 one-hot-vector로 이해할 수 있습니다. 이 점을 고려하여 교차 엔트로피 오차의 식을 살펴보면, 실제값의 차원 이외에서는 모두 0의 값을 갖는 것을 알 수 있습니다. 따라서, 이진 교차 엔트로피 오차와 비슷하게 실제값과 해당 범주의 예측값이 상이할 경우 높은 오차를 갖게 됩니다.

3.2. 역전파(Back Propagation)

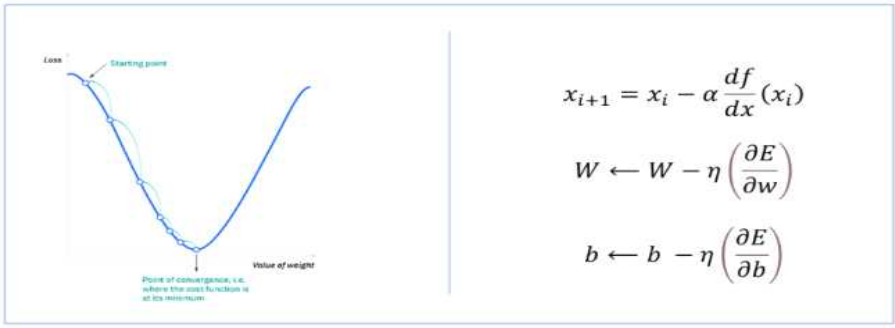
역전파란 모델이 학습을 해 나가는 과정입니다. 모델이 순전파를 진행하는 것을 모델이 추론해 나가는 과정이라고 했는데, 우리는 이 과정을 거치고 나온 예측값과 손실함수를 이용해 정답과 예측의 차이를 구할 수 있습니다. 모델의 학습 목적은 이 차이를 줄이는 것이고 이는 가중치와 편향을 업데이트하는 방식으로 이루어집니다. 업데이트 방식에는 여러가지가 있는데 기본적인 것들에 대해 다뤄보도록 하겠습니다.

3.2.1. Optimizer

딥러닝 모델에서 가중치와 편향을 업데이트하는 과정은 가중치와 편향을 손실함수에 대해 최적화(Optimization)하는 것으로 볼 수 있습니다. 이러한 최적화를 수행하는 여러가지 방법들을 통틀어서 Optimizer라고 부릅니다. 이런 optimizer에는 다양한 종류가 있는데, 그 중 가장 기본이 되는 경사하강법(Gradient Descent)부터 확률적 경사 하강법(Stochastic Gradient Descent), Momentum, AdaGrad에 대해 알아보도록 하겠습니다.

● 경사하강법

경사 하강법은 영어로 Gradient Descent라고 합니다. 여기서 gradient는 고차원 벡터의 편미분 값을 의미하며, descent는 하강을 의미합니다. 따라서, 경사 하강법은 기본적으로 벡터의 편미분 값, 즉 기울기를 작게 만드는 방향으로 나아가는 형태의 최적화 방법을 의미합니다.



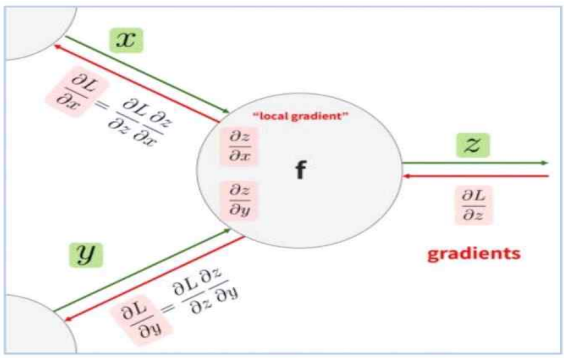
$$x_{i+1} = x_i - \alpha \frac{df}{dx}(x_i)$$

$$W \leftarrow W - \eta \left(\frac{\partial E}{\partial w} \right)$$

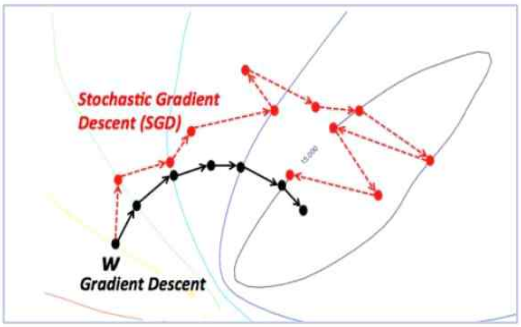
$$b \leftarrow b - \eta \left(\frac{\partial E}{\partial b} \right)$$

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 0에 수렴해갈 것입니다. 이때 미분계수에 비례하여 점이 다음 시점까지 움직일 거리를 정하기 위해 학습률(learning rate, α)이라는 개념을 사용합니다. 만약, 업데이트되는 가중치가 1개라면 오른쪽의 수식으로 업데이트가 가능하겠지만, 딥러닝 모델의 수많은 가중치를 층별로 업데이트하기 위해서는 가중치를 벡터의 형태로 놓고 편미분을 해야합니다. 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. 따라서, 미분의 연쇄법칙을 통해 손실함수의 값을 최종적으로 입력까지 전달하게 됩니다. 이는 그림을 통해서만 확인해보고 넘어가도록 하겠습니다.

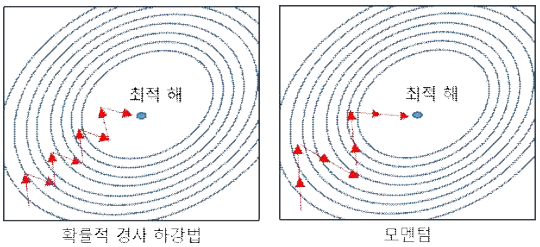


● 확률적 경사 하강법 (Stochastic Gradient Descent)



확률적 경사 하강법은 실제 딥러닝 모델 학습에서 이후 배울 Adam이라는 optimizer와 함께 가장 자주 사용되는 optimizer입니다. 딥러닝 모델의 학습 데이터는 용량이 매우 크고 데이터의 수 또한 매우 많은 경우가 빈번하기 때문에, 모든 데이터를 활용하여 경사 하강법[4]을 통한 학습을 진행하면 학습의 효율이 떨어질 수 있습니다. 따라서, 확률적 경사 하강법은 전체 데이터를 쪼개서 그 중 일부를 사용하는 경사 하강법을 의미하는 optimizer입니다.[5] 그림에서도 볼 수 있듯이 모든 데이터를 사용할 때보다 효율성과 속도의 측면에서는 일반적인 경사 하강법보다 훨씬 뛰어납니다. 다만 변동이 심하고 우리가 원하는 최소점에 정확히 도달하지 못한다는 단점이 있습니다.

● Momentum



확률적 경사 하강법 모멘텀

$$V(t) = m \times V(t-1) - \eta \frac{\partial}{\partial w} Loss(w)$$
$$W(t+1) = W(t) + V(t)$$

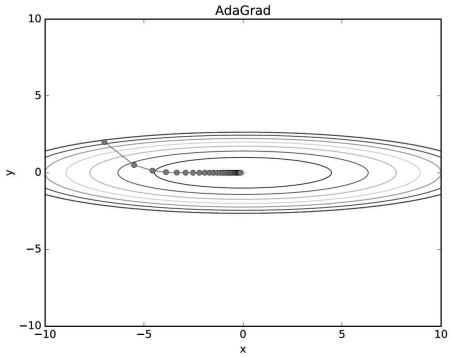
Momentum은 단어 자체의 뜻처럼 관성을 의미합니다. 관성은 진행하던 방향으로 계속해서 진행하려는 성질을 의미합니다. 이를 최적화 문제에 적용하여 생각해보면, 기존 optimizer에서는 미분값에 따라 한 단계 한 단계 최적점에 다가갔다면, 이전 단계의 미분값이 클 경우 일종의 가속도를 부여하여 더 큰 보폭으로 움직이는 것을 의미합니다. 오른쪽 Momentum 수식은 경사하강법 식에 관성항만 추가된 것으로, 다음 시점의 가중치인 $W(t+1)$를 업데이트하기 위해서 이전 시점($t-1$)의 미분값을 사용하는 것을 확인할 수 있습니다. 이는 딥러닝 모델을 학습할 때 자주 마주하게 되는 local minima 문제를 해결하는 데 큰 도움이 될 수 있습니다. 다만, 기울기가 너무 가파른 곳에서 관성항이 너무 커져서 최소 지점을 지나쳐버리는 Overshooting 문제가 발생할 수 있습니다. 해당 문제를 개선한 Nestrov Momentum은 미분값을 업데이트되는 지점에서 계산합니다.

---

4) 이를 일괄 경사 하강법(Batch Gradient Descent)라고 부르기도 합니다.
5) 따라서 확률적 경사 하강법을 Mini Batch Gradieng Descent라고도 합니다.

● AdaGrad

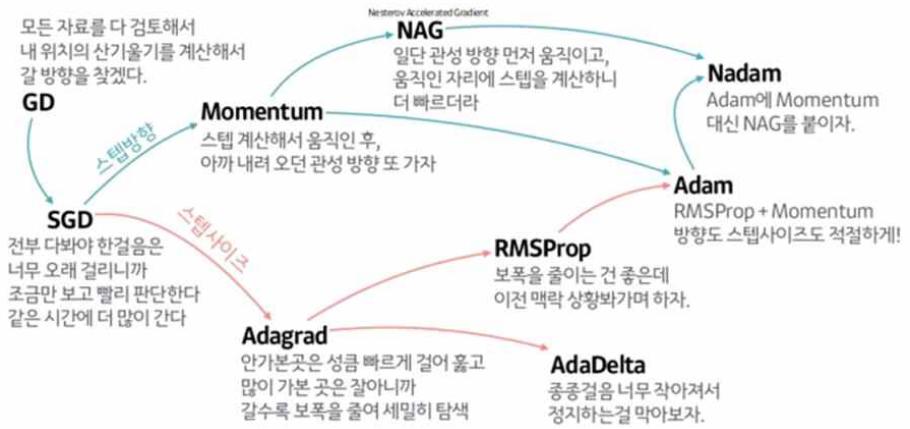
AdaGrad는 Adaptive Gradient의 약자로 가중치 업데이트에 따라 학습률을 조절합니다. 아래의 그림의 y축 방향으로 기울기가 커서 처음에는 크게 움직이게 됩니다. 여기서 크게 움직인 만큼 해당 정보가 이후에 반영되어 점점 작은 폭으로 움직이는 것을 알 수 있습니다. 수식을 보면, 이전의 gradient들의 제곱합이 업데이트식의 학습률에 나누어져, 기존에 크게 업데이트되었다면 점점 적게 업데이트될 것이라고 알 수 있습니다. ϵ의 경우 분모가 0이 되는 것을 방지하는 매우 작은 상수입니다.



$$G(t) = G(t-1) + \left(\frac{\partial}{\partial w(t)} Loss(w(t))\right)^2$$
$$= \sum_{i=0}^{t} \left(\frac{\partial}{\partial w(i)} Loss(w(i))\right)^2$$
$$W(t+1)$$
$$= W(t) - \eta \times \frac{1}{\sqrt{G(t)+\epsilon}} \times \frac{\partial}{\partial w(t)} Loss(w(t))$$

해당 조절은 가중치별로 다르게 적용되기 때문에 효율적으로 최솟값에 수렴할 수 있습니다. 그러나, 학습이 오래 진행될수록 $\frac{1}{\sqrt{G(t)+\epsilon}}$의 값이 0에 수렴하기 때문에 학습이 충분히 이루어지지 못하기 학습이 조기 종료될 수 있는 단점을 가지고 있습니다. 해당 문제를 해결한 RMSProp은 지수이동평균을 사용하여 해당 값이 무한히 작아지는 것을 막아줍니다.

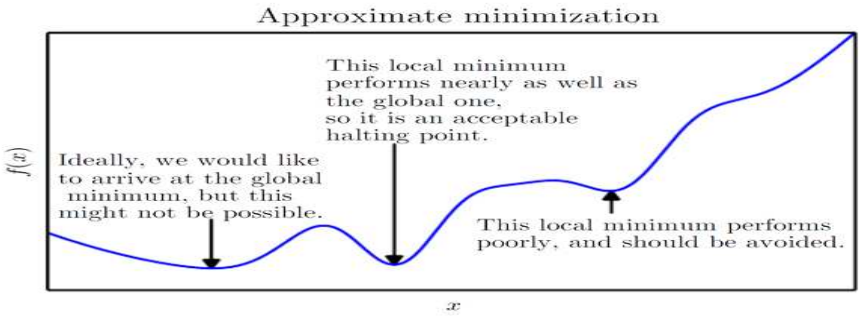
optimizer는 각각의 장점이 있지만 단점도 존재하기 때문에, 이를 해결하기 위해서 새로운 Optimizer들이 새로 나오고 있습니다. 아래 나오는 그림을 통해 여러 Optimizer들을 확인할 수 있습니다. 딥러닝에서 가장 많이 쓰이는 Adam은 아래의 그래프처럼 Momentum과 RMSProp의 특징을 합친 Optimizer입니다. 여기 있는 모든 Optimizer를 기억할 필요는 없고, SGD와 Adam정도만 기억해주시면 좋을 것 같습니다.



3.2.2. Local Minima와 Saddle Point 문제

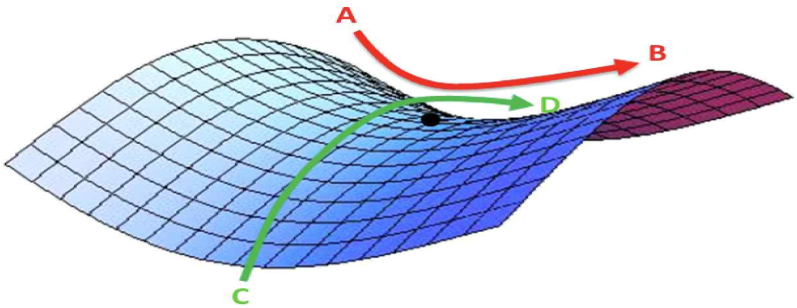
3.2.1 장에서 Optimizer의 단점들이 존재하기 때문에 이를 해결하기 위해 새로운 optimizer가 나왔다고 이야기했는데, 여기서는 그 대표적인 단점들에 대해서 이야기해보자 합니다. 그것들이 바로 Local Minima와 Saddle Point 문제입니다. 각각에 대해서 한번 알아보도록 하겠습니다.

● Local Minima



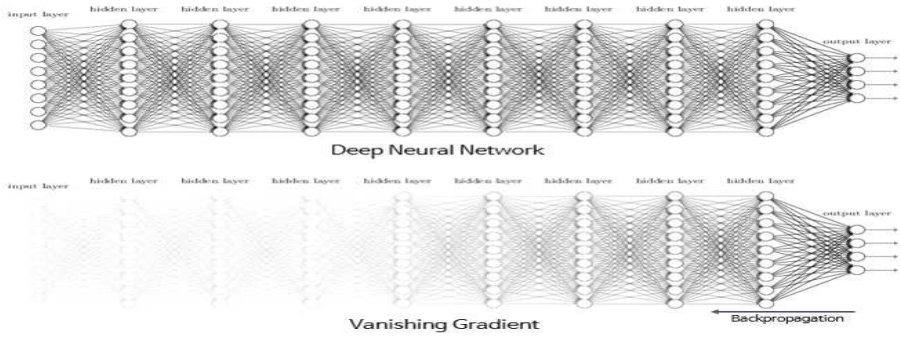
Optimizer를 활용한 최적화 방법들의 목표는 손실함수의 값이 최소가 되는 지점을 찾는 것이었습니다. 하지만, 함수에는 극소 지점이 여러 곳에 존재할 수 있습니다. 따라서 경사 하강법을 활용하여 최소 지점을 찾아가는 과정에서 미분계수가 0이라면 더 이상 학습이 이루어지지 않을 것입니다. 이것이 바로 Local Minima 문제입니다. 이전에 미분계수에 따라 나아가는 방향에 가속도를 적용하는 momentum 방법이 local Minima를 해결할 수 있는 방법이 될 수 있다고 말한 이유도 이것에 있습니다.

● Saddle Point 문제



이 문제는 고차원 함수에서 나타날 수 있습니다. Local minima문제와 비슷하게 Saddle point에서는 미분계수가 0이기 때문에, 더 이상 학습이 이루어지지 않고 멈추는 문제가 있습니다.

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, 0을 중심으로 조금만 값이 벗어나면 미분값이 금방 0에 수렴해버리는 모습을 보입니다. tanh 함수의 경우 미분값의 최고점이 시그모이드 함수에 비해서는 크지만, 0으로부터 멀어졌을 때 미분값이 0에 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 0에 수렴해갈 것입니다. 따라서 최적점과 현위치의 차이를 미분계수가 적절하게 반영해주지 못하게 되어 학습이 멈추게 됩니다.

마지막 그래프에 해당하는 ReLU 함수의 경우 미분값의 그래프가 0 또는 1의 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 딥러닝 모델들에서 가장 대표적으로 활용되는 이유이기도 합니다.

4. 성능 향상 기법

4.1. 가중치 초기화 (Weight Initialization)

0주차의 뉴턴법에서 보았다시피 초기 값을 어떻게 설정하냐에 따라 수렴 속도가 다른 것을 확인할 수 있었습니다. 마찬가지로 신경망도 파라미터를 최적화하는 과정에서, 동일하게 경사 하강법을 통해 내려간다고 하더라도 초기 가중치에 따라서 도달하는 최적점은 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 초기화라 부릅니다.

● zero initialization

우리가 프로그래밍을 할 때 보통 변수를 0으로 초기화를 많이 합니다. 마찬가지로 zero initialization은 파라미터의 가중치를 0으로 초기화시키는 것을 의미합니다. 하지만 파라미터의 값이 모두 같다면 역전파를 통해서 갱신했을 때 같은 값으로 변하게 됩니다. 따라서 이 방법은 딥러닝에서 좋은 방법은 아닙니다.

● Random initialization

파라미터의 값을 정규분포 혹은 균일분포를 따르는 랜덤으로 설정하는 것입니다. 하지만 평균이 0이고 표준편차가 1인 정규분포를 따르게 파라미터를 초기화시킨다면 sigmoid가 활성화함수일 때 input값이 조금이라도 크다면 layer를 통과할수록 그 출력 값이 0, 1에 치우치게 됩니다. 그에 따라 그 지점에서의 gradient도 0에 가까워집니다. Gradient Vanishing 문제가 발생한다는 것입니다. 또한, 평균이 0, 표준편차가 0.01인 정규분포를 따를 때는 그 출력값이 0.5로 치우치게 되어 Zero initialization과 같이 여러 개의 노드를 사용하는 의미가 없어지게 됩니다.

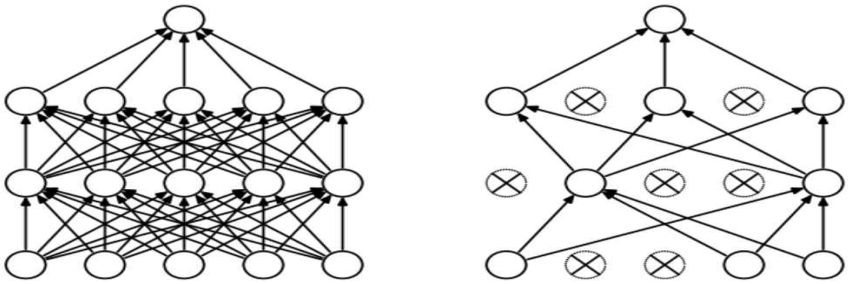
● Xavier initialization

사비에르 초기화는 고정된 표준편차를 사용하지 않고, 이전 은닉층의 노드의 개수가 n개이고 현재 은닉층의 노드가 m개 일때, $\frac{2}{\sqrt{n+m}}$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 initaliaztion

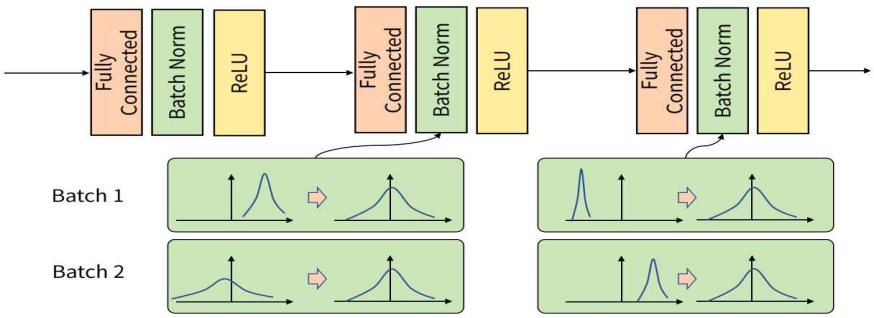
He 초기화는 $\sqrt{\frac{2}{n}}$를 표준편차로 하는 정규분포로 초기화하는 방법입니다. 활성화 함수가 sigmoid 함수일 때는 사비에르 초기화를 많이 사용했지만, 활성화 함수가 ReLU 함수일 때는 층이 깊어지게 될수록 활성값의 분포가 치우치게 됩니다. 이렇게 된다면 Gradient Vanishing 문제가 발생할 수 있습니다. 따라서 활성화 함수가 ReLU 함수일 때는 He 초기화를 많이 사용하게 됩니다.

4.2 Dropout

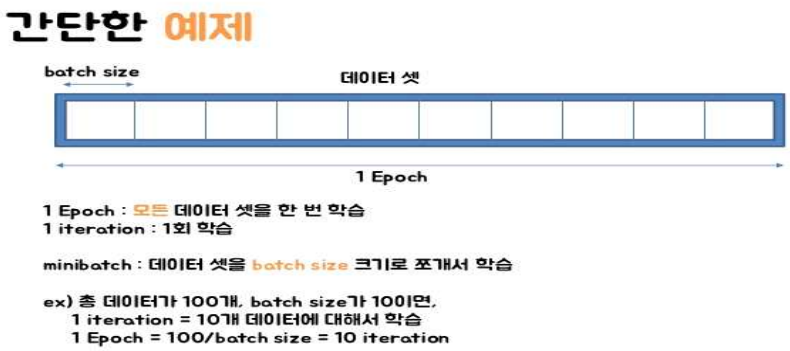


Dropout은 batch마다 학습할 때 일정한 비율의 노드(Node)를 버리고 학습하는 방법을 의미합니다. 이렇게 학습을 진행할 경우, 랜덤으로 일정한 비율의 노드들을 제외하고 학습하기 때문에 여러 모델을 앙상블(ensemble)한 것과 유사한 효과를 낼 수 있습니다. 따라서, 과적합을 방지할 수 있는 방법 중 하나로 이해할 수 있습니다.

4.3 Batch Normalization

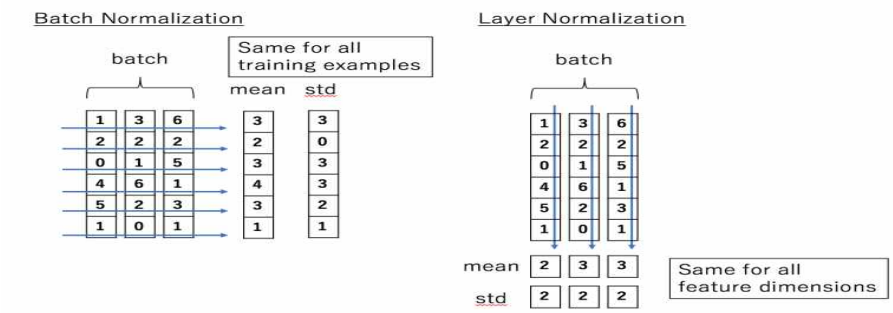


Batch Normalization은 batch마다 다른 분포로 인해 가중치 학습이 batch에 영향을 받는 것을 방지하기 위해 batch별 데이터를 평균과 분산으로 정규화해주는 것을 의미합니다. 보통 각 layer의 활성화 함수 단계 직전에 넣어주게 됩니다. 여기서 batch는 매우 큰 입력 데이터셋을 나누는 단위를 말합니다.



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

학습 과정에서 batch별 평균과 분산을 구하는 것은 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

오늘 1주차에서는 머신러닝의 기본 가정을 통해 딥러닝을 사용하는 이유와 딥러닝의 가장 기본적인 알고리즘을 살펴보았습니다. 딥러닝 또한 머신러닝의 일종이기 때문에 모델, 손실함수, 최적화 기법으로 이루어지며 모델의 경우 다중 퍼셉트론(MLP), 손실함수는 MSE와 교차 엔트로피, 최적화 기법은 경사하강법, 확률적 경사하강법, momentum을 알아보았습니다.

그리고 딥러닝 모델만의 특징이라고 할 수 있는 예측값을 내놓는 과정인 순전파, 순전파에서 나온 결과값을 바탕으로 매개변수를 업데이트하는 역전파 과정을 알아보았습니다. 순전파에서는 딥러닝이 비선형성을 부과하기 위해 활성화 함수를 쓴다는 것, 그리고 그런 활성화 함수는 계단 함수, 시그모이드 함수, ReLU 등이 존재합니다. 그리고 손실함수를 거쳐 역전파를 하게 되는데, 여기서 역전파를 통해 매개변수를 업데이트를 시키는 것의 핵심이 Gradient Descent였습니다. 하지만 이 알고리즘의 문제점이 Local minima나 saddle point에서 학습이 이루어지지 않는다는 것이었습니다.
마지막 4장에서는 이런 딥러닝의 성능을 올리기 위해 가중치 초기화, Drop out, 배치 정규화 등을 다뤘습니다.

다음 시간에는 이론을 바탕으로 간단한 딥러닝 모델을 구현해볼 것이고, 다음 주에는 본격적으로 비정형 데이터, 그 중에서도 이미지를 처리하는 모델을 다루어 보겠습니다. 1주차 고생 많으셨습니다.