4. MLP

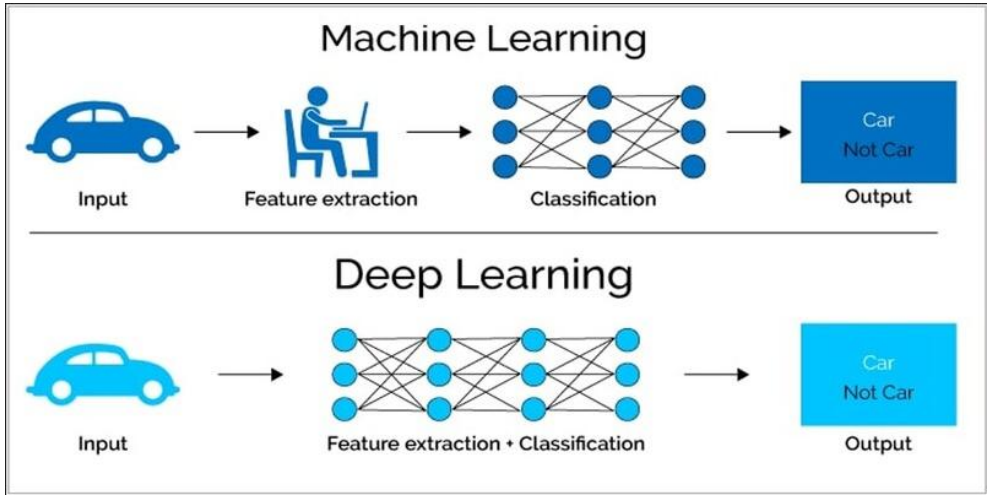
• INDEX

딥러닝

머신러닝

- 컴퓨터에게 데이터를 주고, 그 데이터 안에서 규칙(패턴)을 스스로 학습하게 하는 방식
- 학습 종류
  - 지도 학습(Supervised Learning) => 선형/로지스틱 회귀
  - 비지도 학습(Unsupervised Learning) => K-means, PCA
  - 강화 학습(reinforcement Learning)

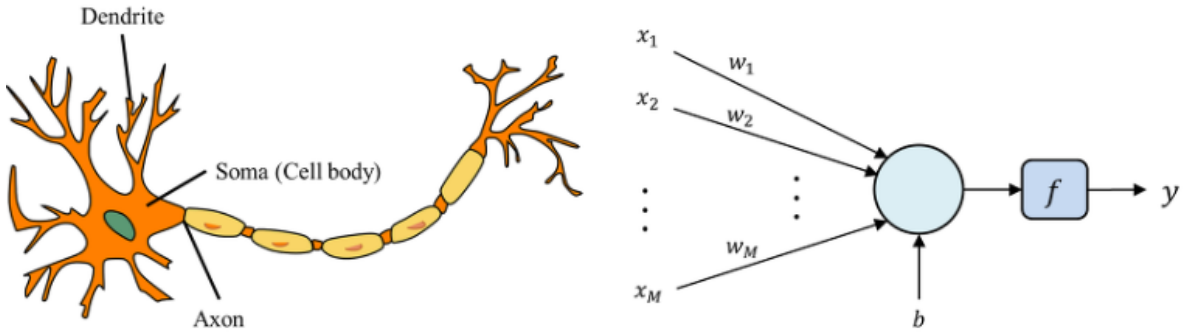
고전적인 머신러닝...



- 고전적인 머신러닝의 한계
  - 모델이 잘 학습할 수 있또록 사람이 직접 데이터의 중요한 부분을 알려주고, 가공해서 먹여줘야 함
  - 데이터의 양이 일정 수준을 넘어서면, 복잡한 패턴을 추가로 학습하지 못하고 성능이 정체 됨
  - 기본적으로 행과 열이 정해진 **정형 데이터 처리**에 적합하여, 이미지나 텍스트, 음성과 같은 "**비정형 데이터**" 처리에 비효율적
- 위 한계를 극복하기 위해서 등장한 방법론이 **"딥러닝"**

딥러닝(Deep Learning) 아이디어

- 인간의 뇌에서 영감을 받은 인공신경망을 사용하여 복잡한 패턴을 학습하는 머신러닝 학습 방법론
- 핵심 아이디어
  - 뇌의 신경망 구조와 동작 방식을 컴퓨터로 모방
  - 뇌가 수십업 개의 신경세포(뉴런)들이 서로 신호를 주고 받으며 학습하고 판단하는 것처럼
  - 컴퓨터 안에 인공뉴런을 만들고, 이들을 연결하여 네트워크를 구축
- 인공 신경망(ANN, Artificial **Neural Network**)
  - 컴퓨터 안에 **인공뉴런(퍼셉트론)**을 만들고, 이들을 연결하여 네트워크를 구축한 것

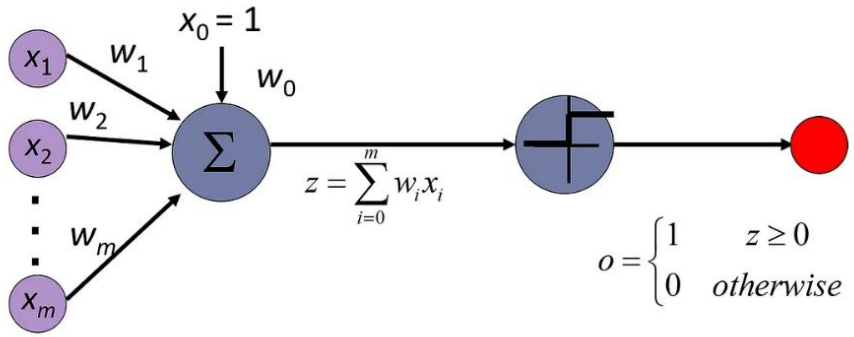


https://velog.io/@palinyee12/DeepLearning-1.-%EC%96%95%EC%9D%80-%EC%8B%A0%EA%B2%BD%EB%A7%9D-part-1

퍼셉트론

퍼셉트론(Perceptron) (1/3)

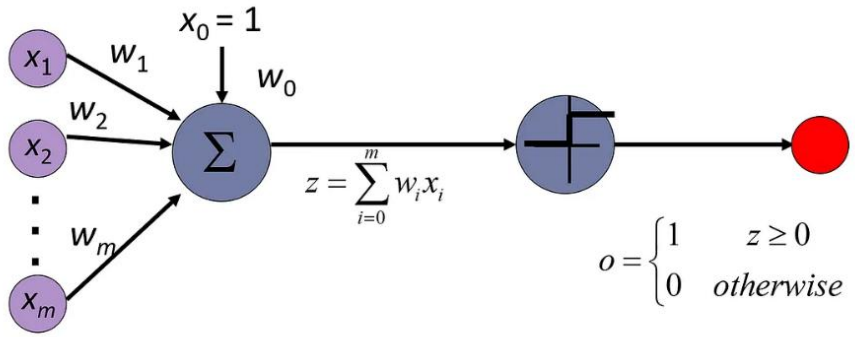
- 여러 정보를 받아서 최종적으로 참(1) 또는 거짓(0)과 같은 결정을 내리는 간단한 결정 모델
- 인공 신경망(ANN)의 가장 기본적인 단위로, 사람의 뇌를 구성하는 뉴런의 작동 원리를 모방한 알고리즘
- 구성 요소
  - 입력(inputs, x)
  - 가중치(weights, w)
  - 가중합(Weighted Sum)
    - 모든 입력에 각각의 가중치를 곱한 뒤, 모두 더한 값 + 편향
    - $(x_1*w_1) + (x_2*w_2) + ... + b$와 같이 "선형 회귀"와 유사
  - 활성화 함수(Activation Function)
    - 입력된 신호를 받아, 그 신호를 처리하여 출력 신호를 결정하는 함수
    - 즉, "활성화" 시켜 전달할지 말지를 결정하는 스위치

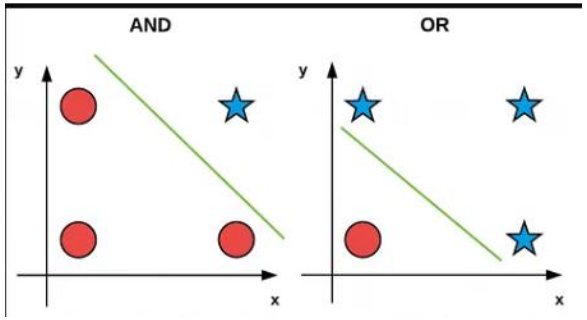


https://ai.plainenglish.io/the-perceptron-a-foundational-building-block-of-neural-networks-6c4bdd29a456

퍼셉트론(Perceptron) (2/3)

- 퍼셉트론은 결국 데이터 사이에 "직선" 하나를 그어서 두 그룹을 나누는 것과 같음
  - 가중합(z) = $(x_1*w_1) + (x_2*w_2) + \ldots + b$
  - z가 0보다 크면 1, 0보다 작으면 0으로 분류
  - 즉 $z = (x_1*w_1) + (x_2*w_2) + \ldots + b$ 가 되는데, 수학적인 이유로 1차 함수를 의미하게 됨
- 그리고 퍼셉트론은 "**선형 분리 가능한 문제**"를 잘 해결했음
- **선형 분리 가능한 문제**
  - 데이터들을 하나의 직선으로 완벽하게 나눌 수 있는 문제
  - 예1) 논리 게이트(AND, OR)
  - 예2) 간단한 분류 문제(스팸 메일 분류, 암 진단)



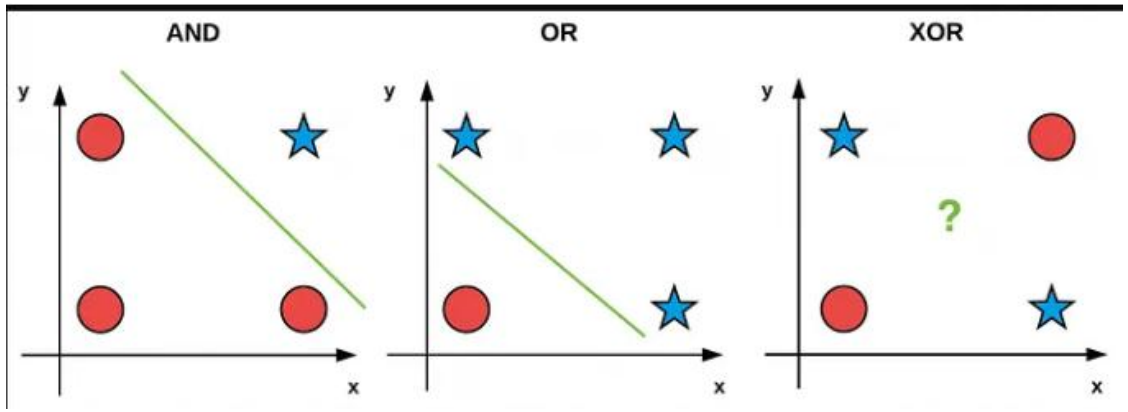


- 기계가 데이터를 보고 스스로 경계선(규칙)을 찾아낼 수 있다는 가능성을 보여준 엄청난 알고리즘

https://medium.com/@amanatulla1606/the-limitations-of-single-layer-perceptron-in-machine-learning-debf0fe959f8
https://ai.plainenglish.io/the-perceptron-a-foundational-building-block-of-neural-networks-6c4bdd29a456

퍼셉트론(Perceptron) (3/3)

- "XOR(베타적 논리합) 문제"를 해결할 수 없는 치명적인 문제를 가지고 있음
  - XOR: 두 입력이 서로 다를 때만 1(참), 같으면 0(거짓)
  - "가격은 싼데(T) 리뷰는 좋거나(T), 가격은 비싼데(F) 리뷰는 안 좋은 곳(F)이 진짜 숨은 맛집(F)이다!"

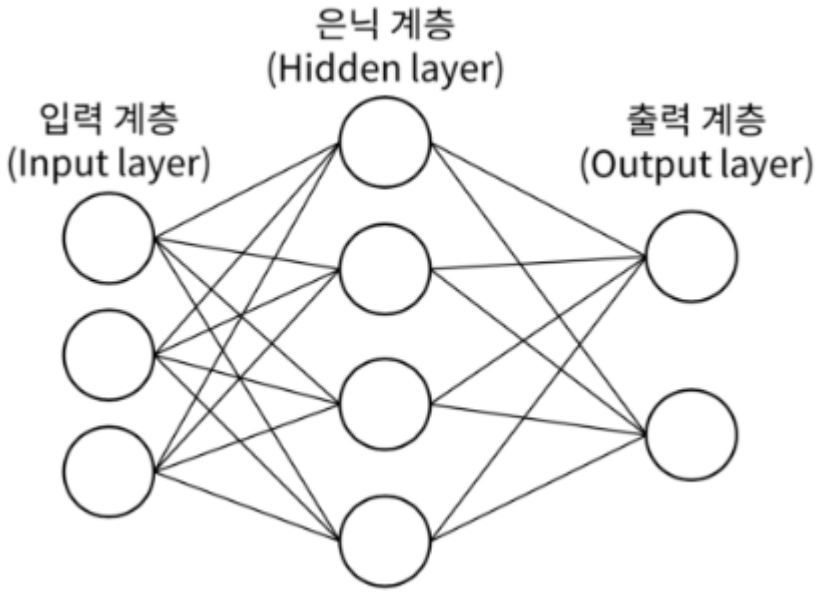


- XOR 문제 때문에 퍼셉트론은 근본적인 한계가 있다는 비판과 함께 AI의 겨울이 도래…
- 하지만, 퍼셉트론을 여러 층으로 쌓아 직선이 하나가 아닌, 여러 개의 직선을 조합하여 XOR 문제를 해결!!!!!
  - ➢ 신경망(Neural Network) 의 시작

https://medium.com/@amanatulla1606/the-limitations-of-single-layer-perceptron-in-machine-learning-debf0fe959f8

Neural Network

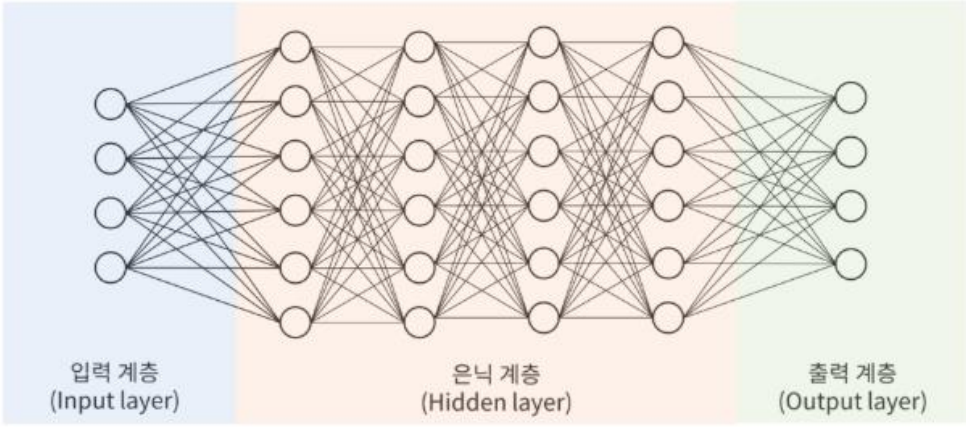
얕은 신경망(Shallow Neural Network, SNN)



- 초기 인공 신경망의 형태
- 얕은 신경망(Shallow Neural Network)
  - 입력/은닉/출력의 3가지 계층으로 이루어짐
  - 은닉층(Hidden Layer)가 딱 하나만 있는 구조
  - **만능 근사 이론**에 의해서 이론적으로는 모든 문제를 해결할 수 있음
- **만능 근사 이론**
  - **이론적으로** 은닉층에 뉴런이 충분히 많다면
    단 하나의 은닉층만으로도 세상의 거의 모든 연속적인 함수를 흉내낼 수 있음
  - 하지만, 은닉층을 터무니 없이 넓게 만들어야했고(계산량 매우큼), 단계별/조합적 학습이 불가능
- 그래서 뉴런을 넓게 만드는 것이 아니라, 깊게 쌓아올리는 아이디어가 "**심층 신경망**"

심층 신경망(Deep Neural Network, DNN) ≈ 딥러닝

- 신경망을 옆으로가 아닌, 깊게(Deep) 쌓는 방식
- 여러 개의 은닉층을 가진 인공 신경망
- 얕은 신경망과 비교
  - 얕은 신경망: 뉴런 12개 => 조합 12개(재사용 불가능)
  - 깊은 신경망: 뉴런 6 개* 6개 => 조합 36개
- 더 복잡하고 고차원적인 특징을 학습 가능
- 사람이 특징을 알려주지 않아도, 복잡한 패턴을 스스로 발견

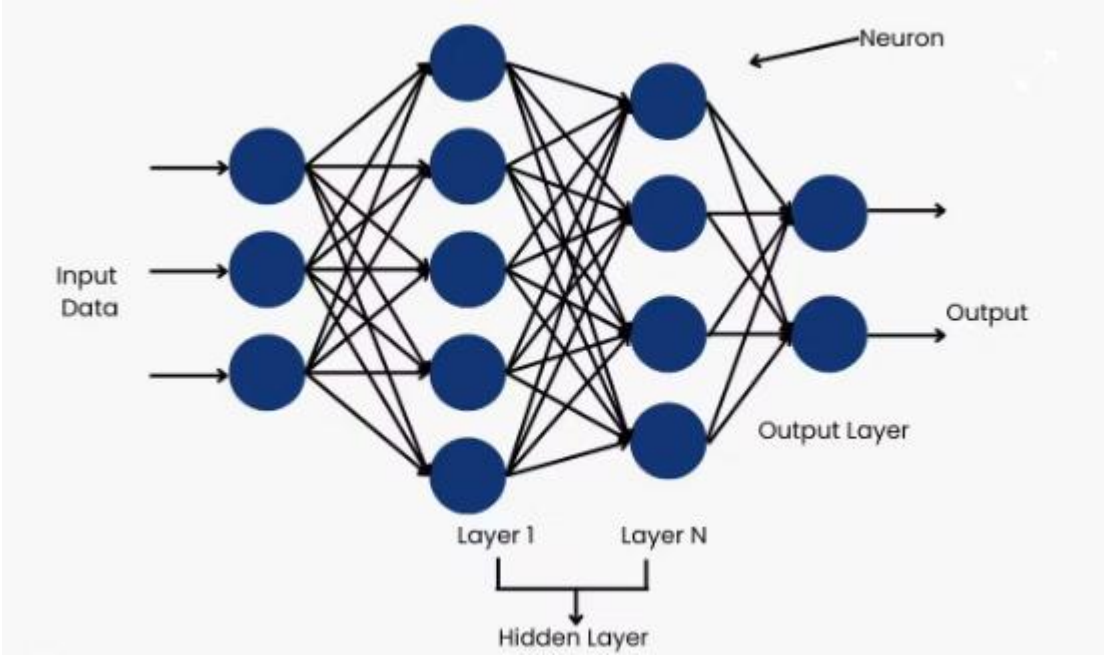


- 정리하자면, 심층 신경망이라는 모델을 사용하여 딥러닝을 진행!

MLP

다중 퍼셉트론(Multi-Layer Perceptron, MLP)

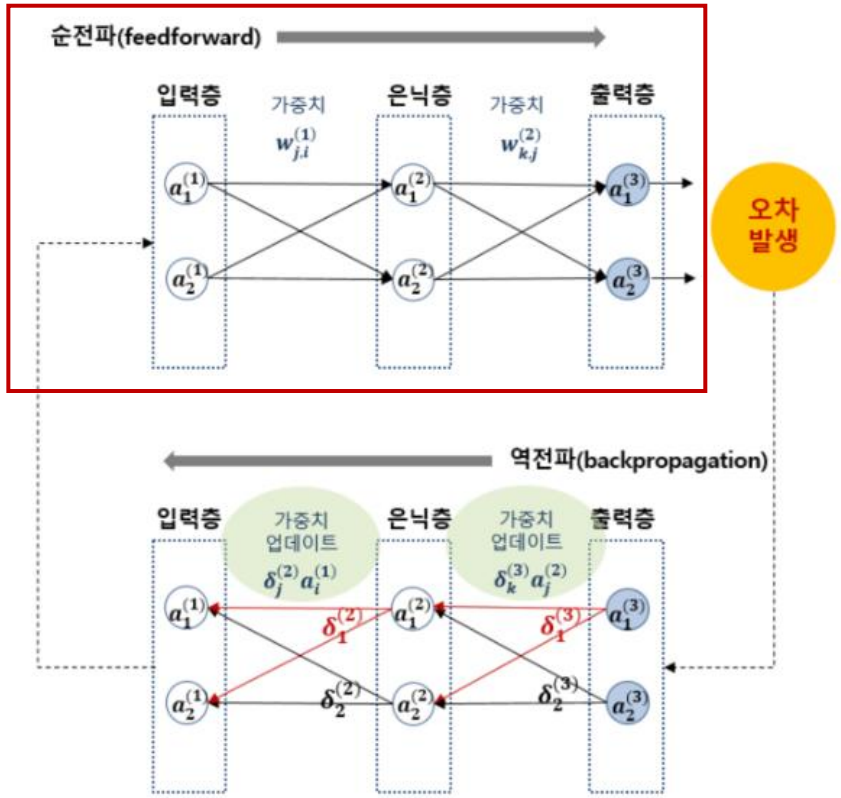
- 인공 신경망(Neural Network) 의 한 종류로, 퍼셉트론을 여러 층으로 쌓은 모델
- 입력층과 출력층 사이에 1개 이상의 은닉층을 가짐
- MLP의 학습 과정
  1. 순전파(Forward Propagation)
  2. 손실 함수 계산(Loss Function)
  3. 역전파(Back Propagation) 및 가중치 업데이트 (Optimization)
  4. 1~3번 반복 후 종료 (Training 완료)



https://www.turing.com/kb/explanation-of-deep-neural-network-multilayer-perceptron-deep-q-network

MLP 학습 과정 - 순전파(Forward Propagation)

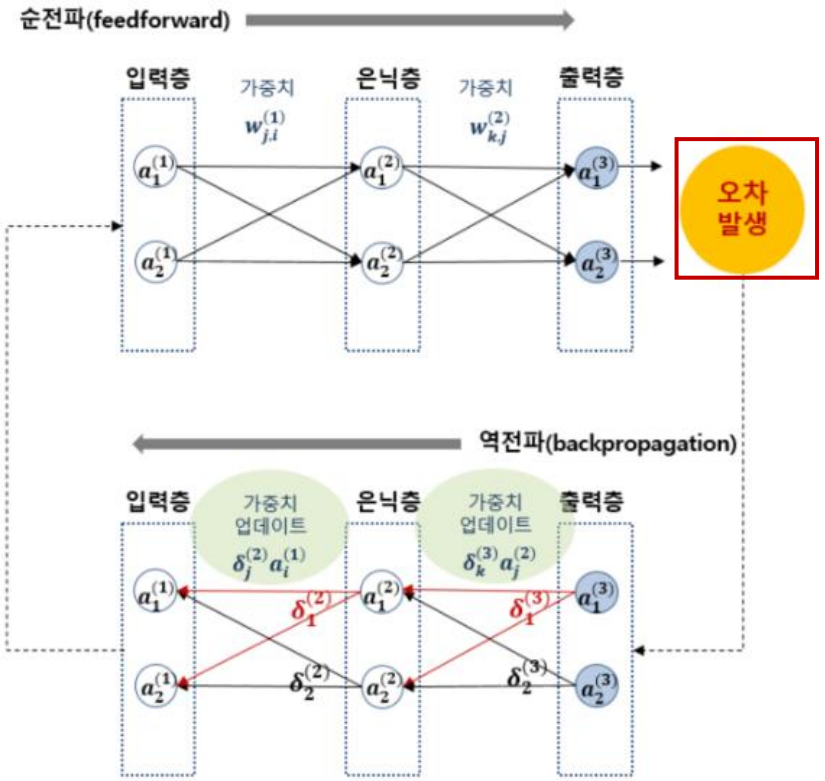
- 데이터가 입력층에서 시작해서 여러 은닉층을 거쳐 출력층까지, 앞에서 뒤로 흘러가면서 계산되는 과정
- 순전파 과정
  1. 초기에 각 뉴런(퍼셉트론)들은 무작위 가중치를 갖음
  2. 데이터(특징 벡터)가 연결된 입력층으로 들어감
  3. 각 뉴런은 입력값과 무작위로 초기화된 가중치를 이용해 계산하고, 활성화 함수(RELU)를 통과시켜 결과를 다음 층으로 전달
  4. 마지막 출력층까지 반복되어 최종 예측값을 출력
- 초기에는 가중치가 랜덤이기 때문에, 매우 엉터리 결과가 나옴



https://wikidocs.net/196164

MLP 학습 과정 - 손실 함수 계산(Loss Function)

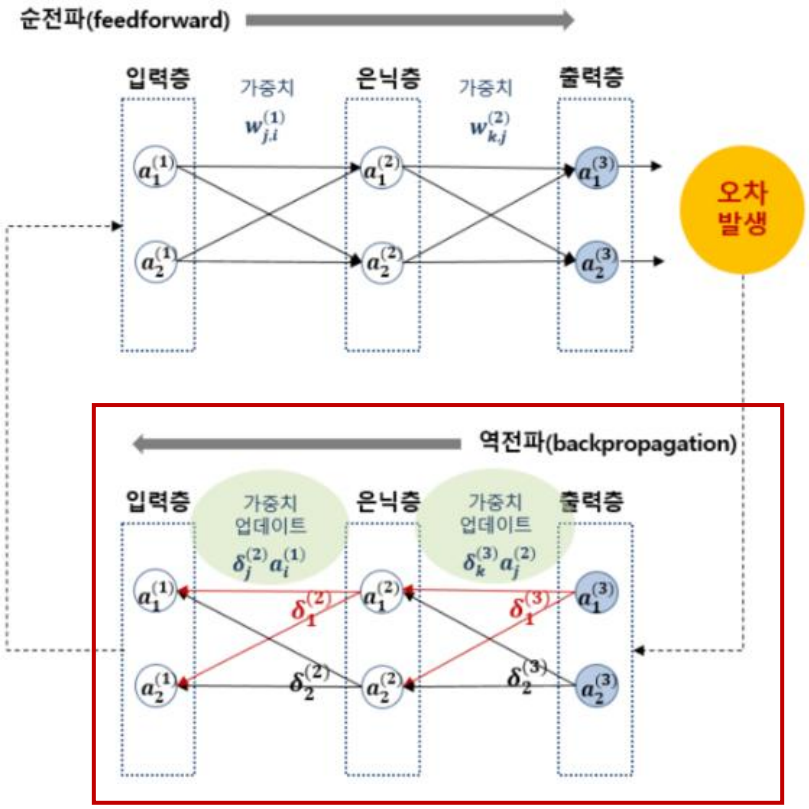
- 순전파 과정에서 나온 결과를 실제 정답과 비교해서 손실(Loss)/오차(Error)를 구함
- 구하는 방법
  - 회귀 (평균 제곱 오차, 각 오차의 제곱의 합)
  - 분류 (교차 엔트로피, 모델이 강하게 확신하고 틀리면 강한 벌점)
  - 비지도(클러스터링 후 떨어진 거리의 총합을 줄이기, 심플 버전)



https://wikidocs.net/196164

MLP 학습 과정 - 역전파(Back Propagation)

- 손실 값을 가지고, 출력층에서 거꾸로 계산하면서 가중치를 업데이트하는 과정
- 역전파 과정(쉬움버전)
  1. 발생한 오차를 가지고, 이전 층으로 거꾸로 추적하면서 각 가중치가 오차에 준 영향을 구함
  2. 계산된 영향력을 바탕으로 오차를 줄이는 방향으로 각 가중치를 조금씩 업데이트
  3. 첫은닉층까지 반복 후 다시 순전파 반복
- 이해가 어려우면 아래 링크를 참고하는 것도 추천
  - https://www.youtube.com/watch?v=tkH7KgLZc0E



https://wikidocs.net/196164

[심화] 역전파 진행 과정 (1/2)

책임 신호: 뉴런의 기울기=영향력

1. 예측값과 정답 차이에서 하나의 "**최종 오차(L)**"가 발생

2. "**최종 오차(L)**"가 뒤로 전파되면서, 각 뉴런은 **최종 오차(L)**에 대한 각기 다른 '책임 신호'를 전달받는다.
   - 책임 신호는 뉴런의 출력값이 **최종 오차(L)**에 얼마나 영향을 미쳤는지를 나타냄
   - 수학적으로는 "**최종 오차(L)**"를 "각 뉴런의 출력"으로 편미분

3. 뉴런 내부의 각 (가중치(w) × 입력데이터(x))은 책임 신호에 각기 다른 영향을 미쳤고,
   이 영향력에 맞춰서 가중치를 수정해야 한다.
   - 영향력이 크면 가중치 수정도 더 많이 해야한다.

[심화] 역전파 진행 과정 (2/2)

책임 신호: 뉴런의 기울기=영향력

최종 책임 신호: 가중치의 기울기

3. 뉴런 내부의 각 (가중치(w) × 입력데이터(x))은 책임 신호에 각기 다른 영향을 미쳤고, 이 영향력에 맞춰서 가중치를 수정해야 한다.
   - 영향력이 크면 가중치 수정도 더 많이 해야한다.

4. (가중치(w)*입력데이터(x))의 영향력에서 실질적으로 책임 신호에 영향을 주는 부분은 "입력데이터(x)"이다.
   - 우리가 궁금한 건 "가중치를 조절했을 때의 결과가 얼마나 변할까? 이고, 이 가중치의 영향력을 결정하는 것은 입력데이터의 몫
   - 예를 들어) 방정식 b=7*a 가 있을 때, a의 영향력을 보고 싶으면 결국 a를 고정하고 7을 쳐다봐야 한다.
   - 그리고 이것이 미분(편미분)이다…!

5. 그렇기에 기존에 있던 가중치(w)는 신경쓰지 말고, 입력데이터(x)과 책임 신호를 곱해서 최종 책임 신호(가중치의 기울기)를 계산 후 학습률과 곱해서 각 가중치 값을 업데이트한다.
   - 입력 데이터가 ↑ -> 영향력이 ↑ -> 가중치의 업데이트 변화량도 커야하고 -> 학습률이랑 곱해질 최종 책임 신호 값도 커야한다.

MLP 학습 과정 - Traning 완료

- 순전파 -> 오차 계산 -> 역전파 과정을 언제까지 진행할까?

1. 정해둔 학습 횟수(Epoch)에 도달
   - 언제 끝날지 예측이 가능하고, 구현이 간단
   - 너무 적게 반복하면 과소적합, 너무 많이 반복하면 과적합
2. 조기 종료(Early Stopping) - 가장 많이 쓰이는 방법
   1. 훈련/검증/테스트 데이터로 나눈 후, 훈련 데이터로 가중치를 업데이트
   2. 1 Epoch가 끝날 때마다, 검증데이터로 오차를 확인
   3. 훈련이 지속될수록 오차가 줄어들다가, 어느 순간 과적합으로 인해 오차가 떨어지기 시작
   4. 여기서 연속으로 N번이상 오차의 변화가 없거나 떨어지면 학습을 중단
   - 모델이 가장 성능이 좋았던 '최적의 순간'에 학습을 멈추게 해줌

MLP 성능 향상

MLP 성능 향상

MLP 성능 향상시키는 기술들

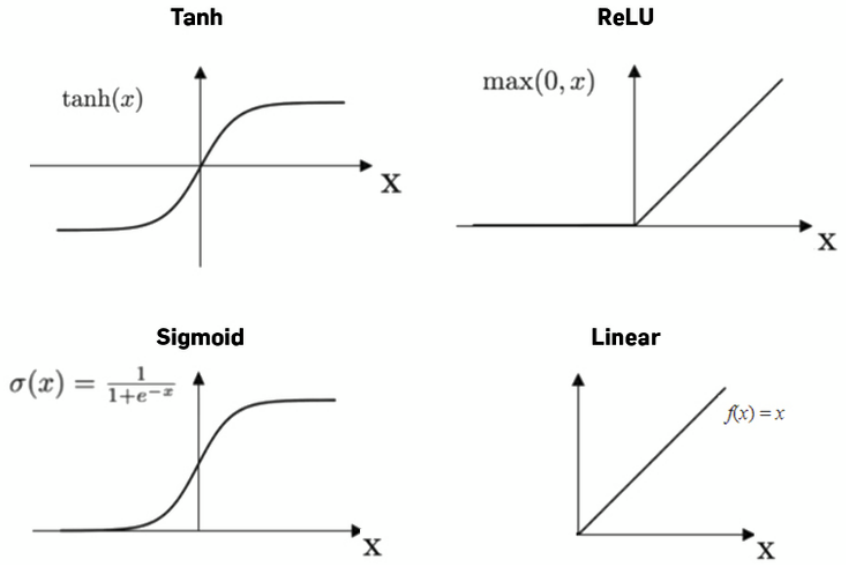
- 우리가 배운 기본 MLP은 실제 복잡한 환경에서 제대로 동작하지 않음
- 대표적인 성능향상 기술
    1. 더 좋은 활성화 함수(ReLU)
    2. 더 효율적인 학습을 위한 옵티마이저(Optimizer)
    3. 과적합(Overfitting)을 막기 위한 규제(Regularization)

활성화 함수(ReLU)

활성화 함수(ReLU)

활성화 함수

- 입력된 신호를 받아, 그 신호를 처리하여 출력 신호를 결정하는 함수
- 즉, "활성화"시켜 전달할지 말지를 결정하는 스위치
- 활성화 함수가 필요한 이유
  - 활성화 함수가 없다면, 신경망은 여러 층을 아무리 깊게 쌓아도 결국 거대한 선형 함수와 다를 바가 없음
  - 활성화 함수는 이런 선형 함수를 훨씬 복잡한 패턴을 학습할 수 있게 함
- 대표적인 활성화 함수
  - Linear : 입력값을 그대로 출력
  - Sigmoid : 0 ~ 1 사이 값으로 출력
  - ReLU: 0보다 작은 값은 0, 0보다 크면 그대로 출력

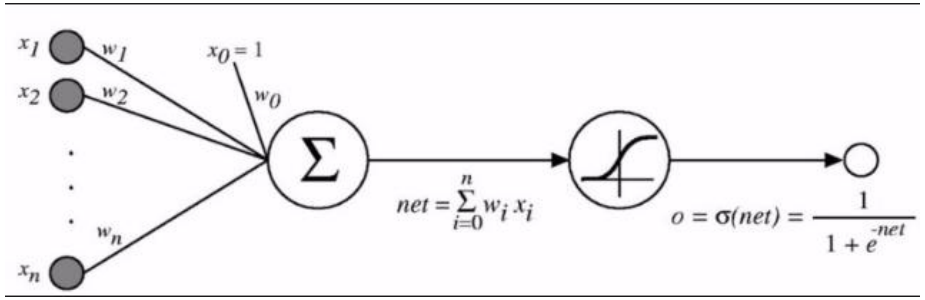


https://machine-learning.paperspace.com/wiki/activation-function

활성화 함수(ReLU)

MLP의 초기 활성화 함수

- 초기 신경망은 시그모이드(Sigmoid) 함수를 활성화 함수로 사용
- 값을 0~1사이로 만들어주니 좋아보였으나, 역전파 과정에서 "기울기 소실 문제(Vanishing Gradient Problem)"가 발생
- **기울기 소실 문제(Vanishing Gradient Problem)**
  - 시그모이드 함수는 역전파 과정에서 '**오차 신호**'를 뒤로 전달할 때마다 미분하는 과정에서 신호의 세기를 아주 약하게 만듦
  - 층을 거꾸로 거슬러 올라갈 때마다, 이 작은 값들이 계속 곱해짐
  - 결국 입력층 근처로 갈수록 0에 가깝게 사라져 버림(Vanishing)
  - 앞쪽 층에 있는 뉴런들은 오차에 대한 피드백을 전혀 받지 못해 학습이 멈춰버림



- 기울기 소실 문제로 인해 수십년 전에 등장한 MLP 이지만, 빛을 보지 못함

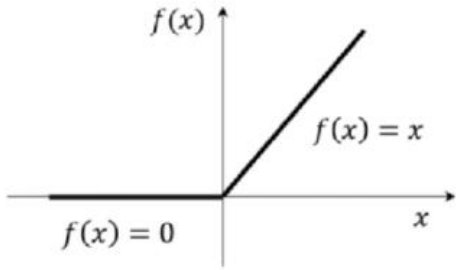
➢ 하지만, 이 때 등장한 것이 ReLU 활성화 함수

https://www.mldawn.com/backpropagation-algorithm-part1-mlp-sigmoid/

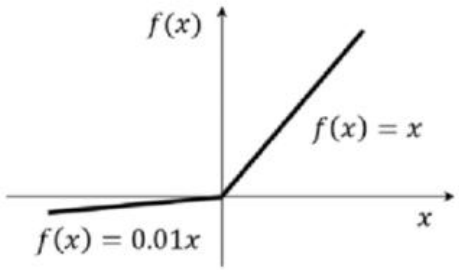
활성화 함수(ReLU)

ReLU 활성화 함수

- Rectified Linear Unit
- 입력값이 0보다 작으면 0, 0보다 크면 값을 그대로 출력
- ReLU가 MLP의 성능을 향상 시킨 이유



ReLU activation function



LeakyReLU activation function

1. 기울기 소실 문제 해결
   - 입력값이 0보다 클 때, ReLU의 기울기는 무조건 1
   - 역전파 과정에서 신호의 세기가 줄어들지 않고 앞쪽 층까지 온전히 전달 됨
   - but 지속적인 음수만 들어오는 경우 생기는 DeadLeLU 문제가 있으며, 이를 해결하기 위해 **Leaky ReLU**를 활용
2. 매우 빠른 계산 속도
   - 시그모이드 함수와 같이 복잡한 지수 함수 계산이 필요없음

https://www.researchgate.net/figure/ReLU-activation-function-vs-LeakyReLU-activation-function_fig2_358306930

옵티마이저(Adam)

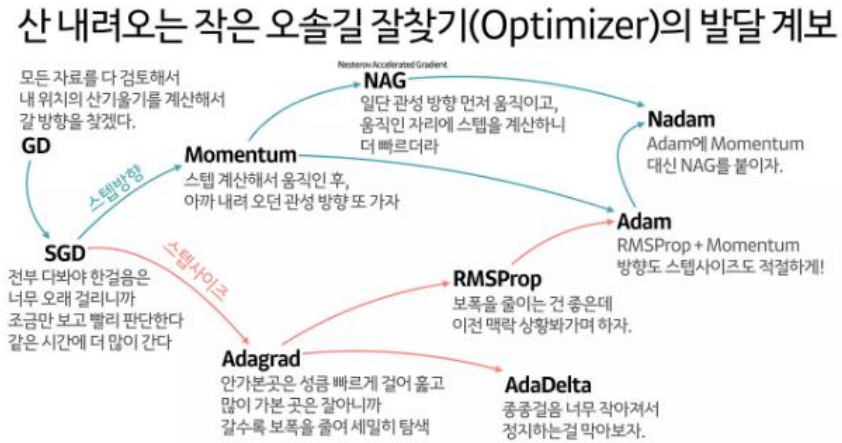
옵티마이저(Adam)

옵티마이저(Optimizer)

- 머신러닝에서 손실 함수의 값이 최소가 되는 최적의 가중치를 찾기 위해 사용되는 알고리즘
- 가장 기본적인 옵티마이저에는 "**경사하강법(Gradient Descent, GD)**"
- 경사 하강법의 치명적인 단점
  - 학습률(Learning Rate) 에 따른 다양한 문제 발생
  - 학습률이 높은 경우: 손실 함수 값이 줄어들지 않고 오히려 커지는 "오버슈팅(overshooting)" 발생
  - 학습률이 낮은 경우: 학습이 너무 느리고, 지역 최저점(Local Minima)에 빠지면 학습을 멈추거나 매우 느려짐

- 이런 경사하강법의 단점을 해결한 방법이

➢ **Adam**(Adaptive Moment Estimation)

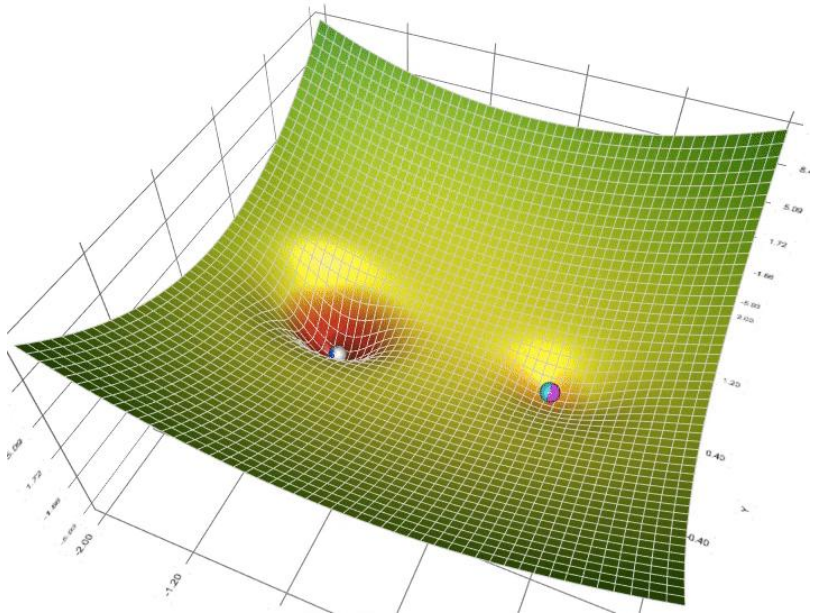


https://coco0414.tistory.com/115

옵티마이저(Adam)

Adam(Adaptive Moment Estimation)

- 현재 딥러닝 모델의 80% 이상이 사용하는 사실상 표준 옵티마이저
- 이름처럼 "적응적(Adaptive)으로 학습률(Learning Rate)를 조절하는 것이 핵심"
- Adam의 아이디어
  1. 관성(Momentum) 도입
  2. 적응적 학습률(Adaptive Learning Rate) 도입



Animation of 5 gradient descent methods on a surface: gradient descent (cyan), momentum (magenta), AdaGrad (white), RMSProp (green), Adam (blue). Left well is the global minimum; right well is a local minimum.

https://medium.com/data-science/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

옵티마이저(Adam)

Adam(Adaptive Moment Estimation)

1. **관성(Momentum) 도입**
   - 현재 위치의 기울기만 보고 움직이는 경사하강법과는 다르게, **이전까지 이동해 온 '속도와 방향(관성)'을 기억**
   - 양쪽 벽으로 요동치는 움직임을 상쇄하고, **관성을 이용해 부드럽게 내려가게 만듦**
   - 잠시 '지역 최저점'에 빠지더라도, **관성의 힘으로 지역 최저점을 빠져나올 수 있음**
2. **적응적 학습률(Adaptive Learning Rate) 도입**
   - 가중치마다 "고유한 학습률"을 부여 하여, 최적의 학습 속도를 유지하여 효율적으로 낮은 손실 지점에 도달
   - **많이** 움직였던 가중치에 대해서는 **학습률을 작게** 만들어서, **적게** 움직이도록 함
   - **적게** 움직였던 가중치에 대해서는 **학습률을 크게** 만들어서, **빠르게** 학습을 진행

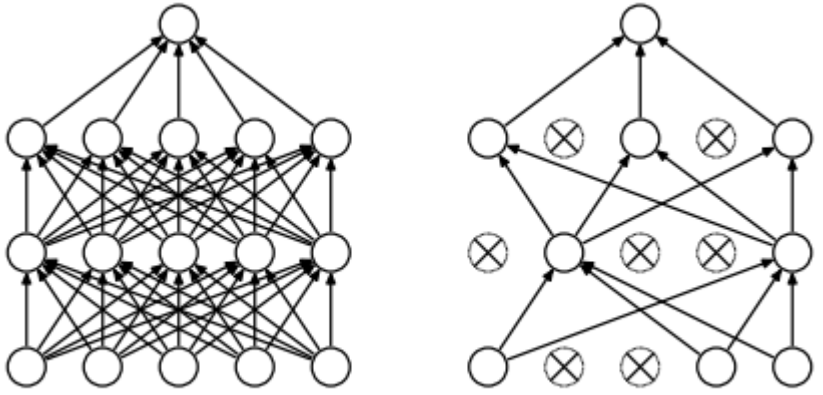
규제(Dropout)

규제(Regularization)

- 모델의 과적합(overfitting)을 막기 위해 제약을 거는 것
- 대표적인 규제 방법
  1. L1 규제 (Lasso)
     - 비용 함수에 L1 norm 을 추가하여 최적화
     - 일부 가중치를 0으로 만들어 제거함으로써 복잡도를 감소시며, 불필요한 특성이 많은 경우 사용
  2. L2 규제 (Ridge)
     - 가중치를 0에 가깝게 만들지만 완전히 0으로 만들지는 않음
     - **다중공선성**이 있을 때, 상관관계가 높은 여러 특성들에 걸쳐 영향력을 고르게 분산시킴
       (다중공선성(Multicollinearity): 두 개 이상의 독립 변수들이 서로 강하게 상관되어 있는 현상)
  3. **드롭아웃(Dropout)**
     - 학습 시 각 뉴런을 랜덤하게 일시적으로 학습시키지 않는 것

드롭아웃(Dropout)

- 학습 시 각 뉴런을 랜덤하게 일시적으로 OFF 시키는 것
- 훈련할 때, 매번 순전파를 진행할 때마다 각 은닉층의 뉴런을 일정 확률(보통 0.5)로 랜덤하게 선택하여 일시적으로 OFF
- 장점
  - 특정 뉴런이 모든 것을 학습하거나(편향), 다른 뉴런에게 지나치게 의존하는 것을 방지
  - 매번 다른 네트워크(약간씩 다른 구조)를 훈련시키는 것과 같은 효과를 냄 (**앙상블 효과**)
- 테스트 시에는 모든 뉴런을 사용해서 테스트를 진행함



(a) Standard Neural Net (b) After applying dropout.

https://blog.skby.net/%EB%93%9C%EB%A1%AD%EC%95%84%EC%9B%83-dropout/