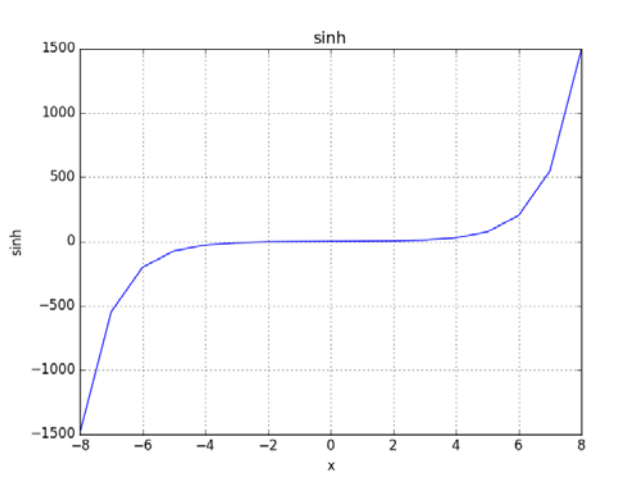
머신러닝 맛보기

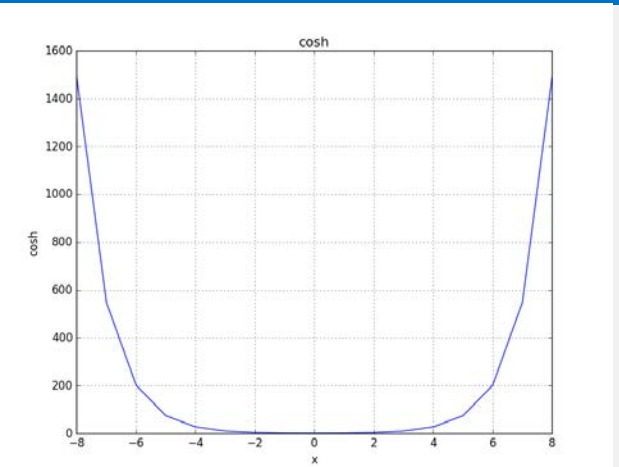
[2주/2차시 학습내용]: 머신러닝 맛보기

-. 인공 뉴런의 동작원리를 미리 봄으로써, 쌍곡선 함수가 머신러닝의 활성화 함수에 사용됨을 알고, 수치해석에서 배우는 수학적 이론이 실제 머신러닝에 활용됨을 확인한다.

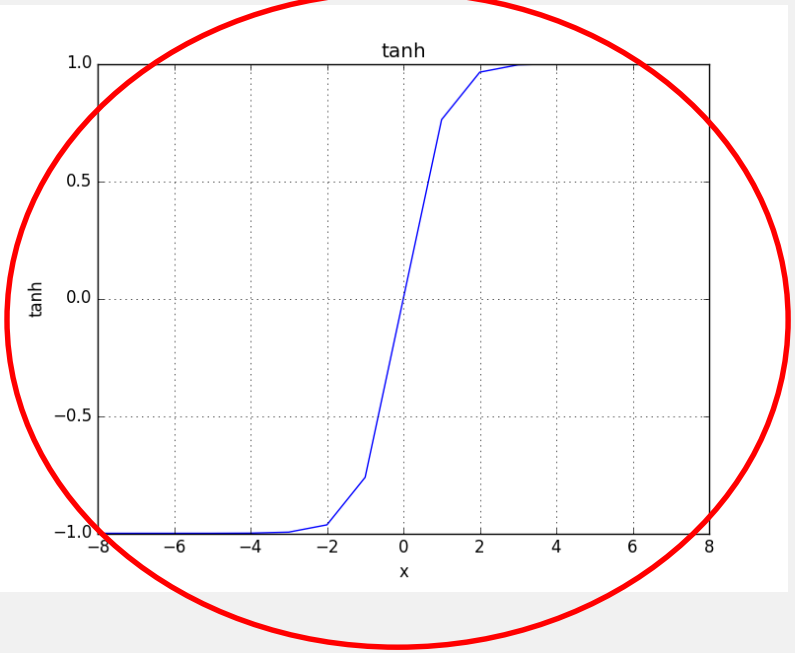
-. 선형회귀를 통해 텐서플로우의 추상화를 접해본다

tanh (쌍곡탄젠트함수)



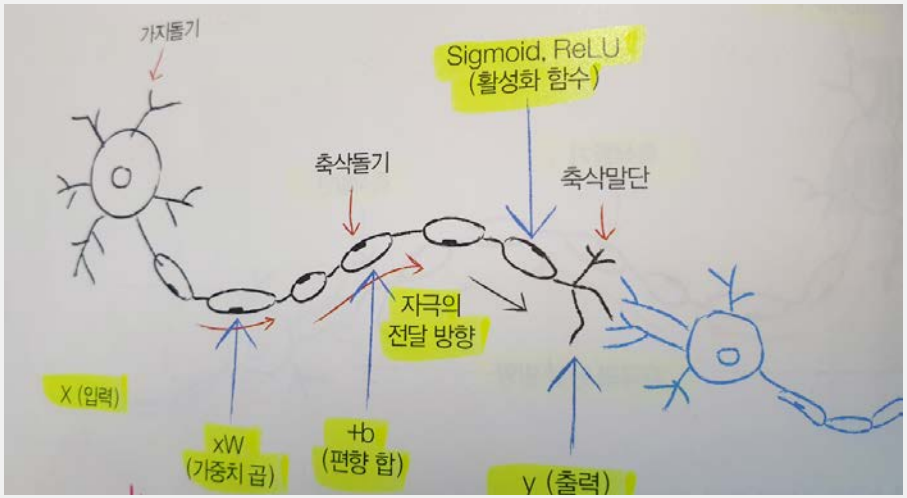



```
import numpy as np
import matplotlib.pyplot as plt
x=np.arange(-8,8+1,1)
y1=np.exp(x)/2
y2=np.exp(-x)/2
y3=y1-y2
y4=y1+y2
y5=y3/y4
plt.figure(1)
plt.plot(x, y5)
plt.xlabel('x')
plt.ylabel('tanh')
plt.title('tanh')
plt.grid(True)
plt.show()
```



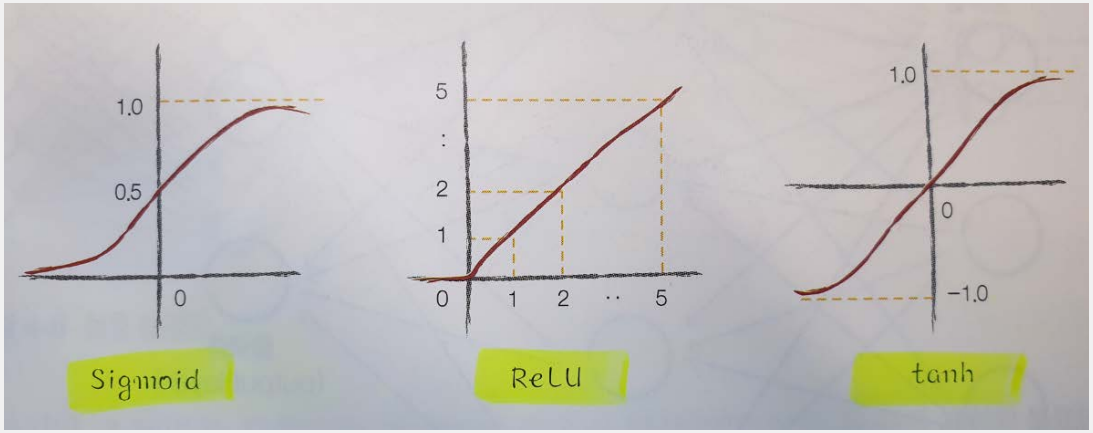
실제 뉴런과 인공 뉴런의 동작원리

- 가지돌기에서 신호를 받아들이고, 이 신호가 축삭돌기를 지나, 축삭단말로 전달된다. 축삭돌기를 지나는 동안 신호가 약해지거나, 강하게 전달되기도 한다. 축삭돌기까지 전달된 신호는 다음 뉴런의 가지돌기로 전달된다.
- 인공 뉴런은 가지돌기에 해당하는 입력값 x부분, 축삭돌기에 해당하는 가중치(w)를 곱하고, 편향(b)를 더하고, 활성화 함수를 거치는 과정, 축삭단말에 해당하는 결과값 y 부분으로 구현된다.
- y=Sigmoid(x*w+b)



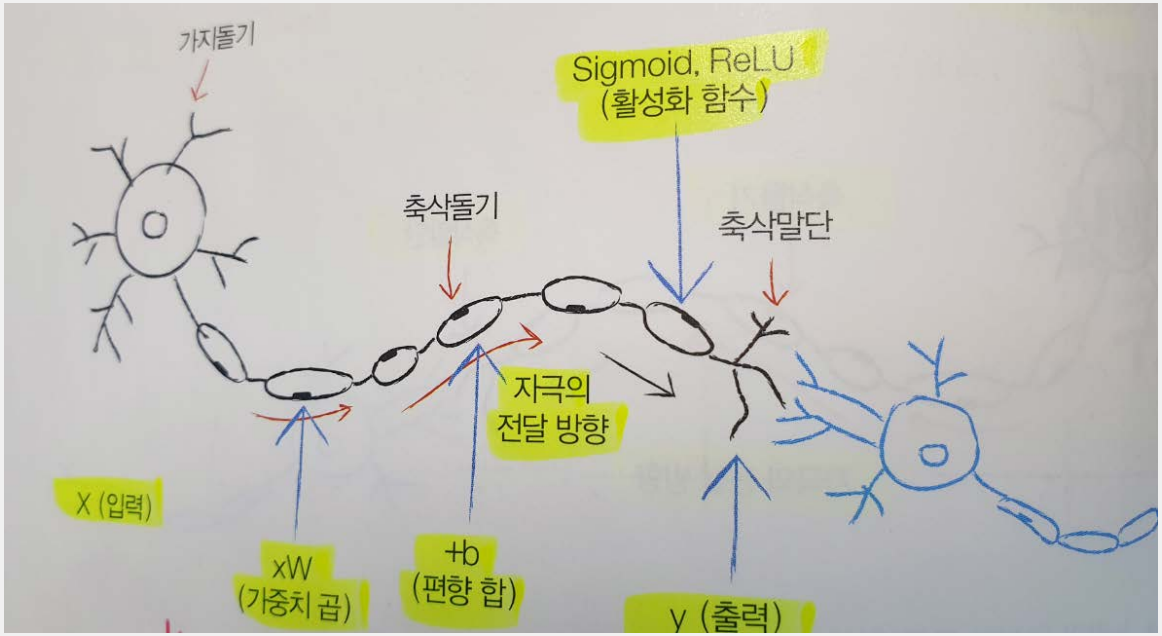
활성화 함수(activation function)와 tanh()

- y=Sigmoid(x*w+b)
  - y: 출력, Sigmoid: 활성화함수, x:입력, w:가중치, b: 편향
  - w:가중치, b: 편향을 찾아내는 것이 학습(Learning)
- 대표적인 활성화 함수는 Signoid (시그노이드), ReLU(렐루), tanh (쌍곡탄젠트) 함수가 있다
- 최근에는 ReLU가 많이 사용되면, 입력값이 0보다 작으면 항상 0을 0보다 크면, 입력값을 그대로 출력한다
- 학습목적에 따라 다른 활성화 함수를 사용



뉴런의 기본동작

- 입력값 x에 가중치(w)를 곱하고, 편향(b)를 더한 뒤 활성화 함수(Signoid (시그노이드), ReLU(렐루), tanh (쌍곡탄젠트))를 거쳐 결과값 y를 만들어 내는 것, 이것이 인공 뉴런의 기본이다.
- y=Sigmoid(x*w+b)



학습 또는 훈련

- 그리고 원하는 y값을 만들어내기 위해 w와 b의 값을 변경해가면서 적절한 값을 찾아내는 최적화 과정을 학습(learning) 또는 훈련(training)이라 한다.
- Signoid 그래프는 0과 1에 한없이 가까워진다.
- ReLU 그래프는 입력값이 0보다 작으면 항상 0을 0보다 크면, 입력값을 그대로 출력한다
- tanh 그래프는 1과 -1에 한없이 가까워진다

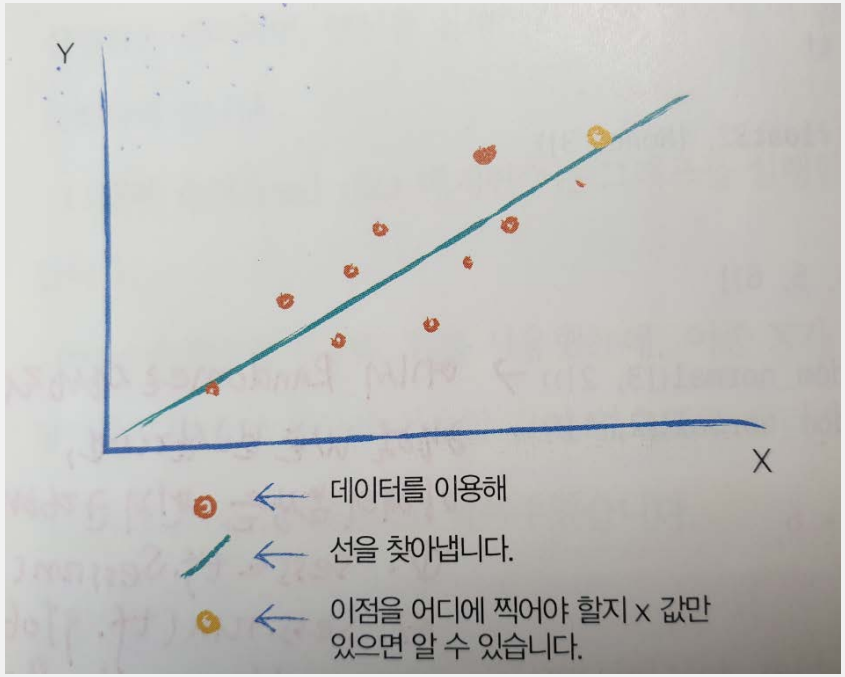
텐서플로우의 추상화 접해 보기

나중에 배울 선형회귀를 통해 텐서플로우의 추상화를 먼저 코딩으로 접해본다

국민대학교 소프트웨어학부

선형회귀 모델 구현해보기

- 선형회귀란 주어진 x와 y값을 가지고 x와 y 간의 관계를 파악하는 것이다.
- 이 관계를 알고 나면, 새로운 x 값이 주어졌을 때, y값을 쉽게 알 수 있다.
- 어떤 입력에 대한 출력을 예측하는 것, 이것이 머신러닝의 기본이다.



플레이스홀더와 변수 개념

- x와 y의 상관관계를 설명하기 위한 변수들인 W와 b를 각각 -1.0부터 1.0사의의 균등분포를 가진 무작위 값으로 초기화

```
import numpy as np
import tensorflow as tf

x_data=[1,2,3]
y_data=[1,2,3]

# x와 y의 상관관계를 설명하기 위한 변수들인 W와 b를 각각 -1.0부터 1.0사의의 균등분포를
가진 무작위 값으로 초기화

W=tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b=tf.Variable(tf.random_uniform([1], -1.0, 1.0))

X=tf.placeholder(tf.float32, name="X")
Y=tf.placeholder(tf.float32, name="Y")

# W와의 곱과 b와의 합을 통해 X와 Y의 관계를 설명하겠다는 뜻이다
# X가 주어졌을 때, Y를 만들어 낼 수 있는 W와 b를 찾아내겠다는 의미이다.
# W:가중치 (weight), b: 편향(bias)
hypothesis=W*X+b
```

W:가중치 (weight), b: 편향(bias)

- W와의 곱과 b와의 합을 통해 X와 Y의 관계를 설명하겠다는 뜻이다
- X가 주어졌을 때, Y를 만들어 낼 수 있는 W와 b를 찾아내겠다는 의미이다.
- W:가중치 (weight), b: 편향(bias)

손실함수(loss function)

- 손실함수(loss function)는 한 쌍(x,y)의 데이터에 대한 손실값을 계산하는 함수이다
- 손실값이란 실제값과 모델로 예측한 값이 얼마나 차이가 나는가를 나타내는 값이다.
- 손실값이 작을수록 그 모델이 X와 Y의 관계를 잘 설명하고 있다는 뜻이며, 주어진 X값에 대한 Y값을 정확하게 예측할 수 있다는 뜻이다.
- 이 손실을 전체 데이터에 대해 구한 경우 비용(cost)라고 한다
- 비용(추상화 함수): `tf.reduce_mean`

```
cost=tf.reduce_mean(tf.square(hypothesis-Y))
```

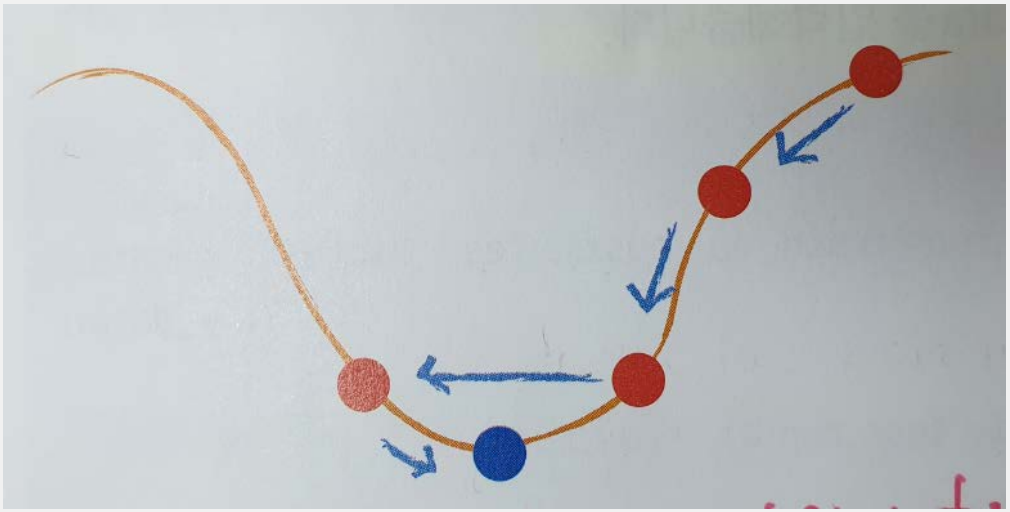
학습이란?

- 학습이란, 변수들(W:가중치 (weight), b: 편향(bias))의 값을 다양하게 넣어 계산해보면서 이 손실값을 최소화하는 W 와 b의 값을 구하는 것이다.
- 손실값으로는 '예측값과 실제값의 거리'를 가장 많이 사용한다.
- 손실값은 예측값에서 실제값을 뺀 뒤 제곱하여 구하며, 그리고, 비용은 모든 데이터에 대한 손실값의 평균을 내어 구한다

경사하강법(Gradient Descent) 최적화 함수

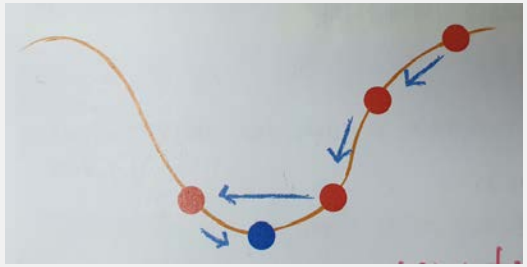
- 텐서플로가 제공하는 경사하강법(Gradient Descent) 최적화 함수를 이용해 손실값을 최소화하는 연산 그래프를 생성한다.

```
optimizer=tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op=optimizer.minimize(cost)
```



경사하강법

- 최적화 함수란 가중치(W)와 편향(b) 값을 변경해가면서 손실 값을 최소화하는 가장 최적화된 가중치(W)와 편향(b)값을 찾아주는 함수이다
- 가중치(W)와 편향(b)값을 무작위로 변경하면 시간이 너무 오래 걸리고, 학습 시간도 예측하기 어렵다
- 빠르게 최적화하기 위한 방법 중의 하나가 경사하강법이다.
- 경사하강법은 최적화 방법 중 가장 기본적인 알고리즘으로 음의 경사 방향으로 계속 이동하면서 최적의 값을 찾아 나가는 방법이다



학습률: 하이퍼파라미터(hyperparameter)

- 학습률은 학습을 얼마나 급하게 할 것인가를 설정하는 값이다.
- 학습률이 너무 크면 최적의 손실값을 찾지 못하고 지나치게 되고, 값이 너무 작으면 학습 속도가 매우 느려진다.
- 학습 진행에 영향을 주는 변수를 하이퍼파라미터(hyperparameter)라 하면, 이 값에 따라 학습 속도나 신경망 성능이 크게 달라진다.
- 머신러닝에서는 하이퍼파라미터를 잘 튜닝하는 것이 큰 과제이다.

학습 수행

- 선형회귀모델을 다 만들었으니, 그래프를 실행해 학습을 시키고, 결과를 확인하자
- 파이썬 with 기능을 이용해 세션 블록을 만들고, 세션 종료를 자동으로 처리하자
- 최적화를 수행하는 그래프인 train_op를 실행하고, 실행 시마다 변화하는 손실값을 출력한다
- 학습은 100번 수행하면, feed_dict 매개변수를 통해, 상관관계를 알아내고자 하는 데이터인 x_data와 y_data를 입력한다.

학습 수행

- 최적화가 완료된 모델에 테스트 값을 넣고 결과가 잘 나오는지 확인해봅니다.

```
with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())

    for step in range(100):
        _, cost_val=sess.run([train_op, cost], feed_dict={X:
x_data, Y: y_data})

        print(step, cost_val, sess.run(W), sess.run(b))

    print("\n=== Test ===")
    print("X: 5, Y:", sess.run(hypothesis, feed_dict={X: 5}))
    print("X: 2.5, Y:", sess.run(hypothesis, feed_dict={X:
2.5}))
```

학습 진행 상황 출력 (손실값과 변수들의 변화 확인)

- 스텝, 손실값, [W 값], [b 값]
- 0 0.869386 [0.9130715] [0.3043009]
- 1 0.02205333 [0.87248445] [0.27821213]
- 2 0.011377501 [0.88021415] [0.27357593]
- 3 0.010722056 [0.8825839] [0.26677507]
- 4 0.010211366 [0.8854622] [0.2603865]
- 5 0.009726319 [0.8882096] [0.2541243]
- 6 0.009264297 [0.8908976] [0.24801561]
- 7 0.008824232 [0.89352024] [0.24205346]
- 8 0.008405092 [0.89607996] [0.23623468]
- 9 0.008005846 [0.8985781] [0.23055576]
- 10 0.00762555 [0.90101624] [0.22501338]

학습 테스트

- 스텝, 손실값, [W 값], [b 값]
- 99 0.000100282195 [0.98864883] [0.02580387]
- 학습에 의해 x=[1,2,3], y=[1,2,3]을 만드는 W 값은 1에 가까운 0.988이 정해졌고, b 값은 0에 가까운 0.025가 정해졌다.
- Test에서는 정해진 [W 값], [b 값] 을 가지고, X: 5가 들어오면 Y 값을 예측하는 데, Y: [4.969048]로 예측하였다. (5에 가까움)
- X: 2.5가 들어오면 Y 값을 예측하는 데, Y: [2.4974258]로 예측하였다. (2.5에 가까움)
- === Test ===
- X: 5, Y: [4.969048]
- X: 2.5, Y: [2.4974258]