Part II
Deep Networks: Modern Practices

Chapter 6

박달님 04.08.21

Definition

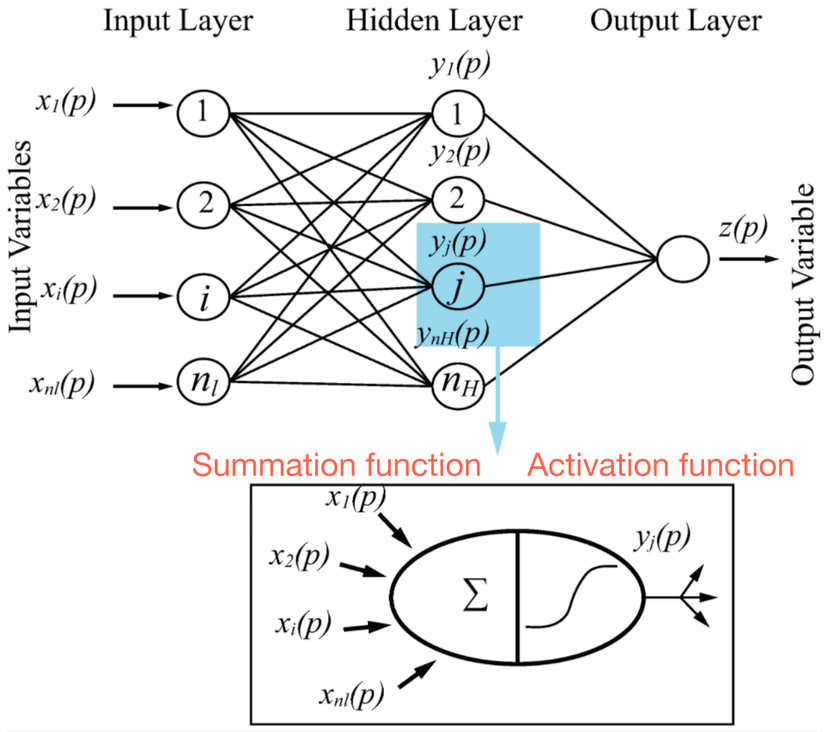
- Deep feedforward networks / feedforward neural networks / MLP
  - Mapping $y = f(x;\theta)$, learns value of parameter θ -> best function approx.
- 고안된 인공 신경망의 최초이자 가장 단순한 유형
- 입력 노드에서 숨겨진 노드를 통해 한 방향으로만 앞으로 이동
- 네트워크에 순환이나 루프가 없다

Model Architecture

$$O_k = g_2\left[\sum_{j=1}^{M} W_{kj}\, g_1\left(\sum_{i=1}^{N} W_{ji} x_i + W_{jo}\right) + W_{ko}\right]$$

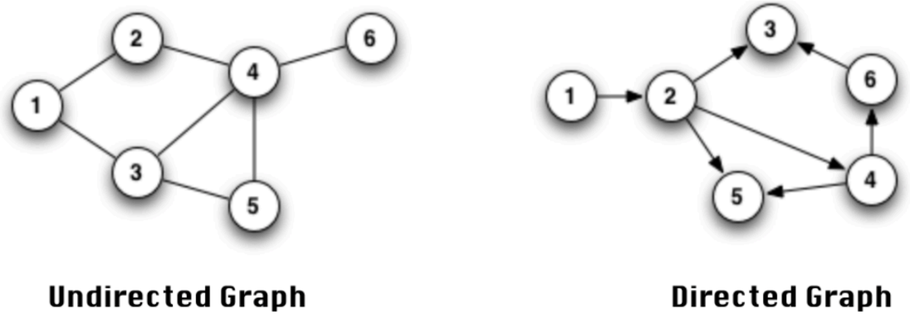
Xi -> node i의 input value
Ok -> node k의 ouput
G1 -> hidden layer의 activation function (nonlinear)
G2 -> output layer의 activation function (linear)
N, M -> input and hidden layers에 있는 뉴런의 개수
Wj0 and Wk0 -> hidden layer의 j번째, output layer의 k번째 뉴런의 biases
Wji -> input node i와 hidden node j 사이의 weight
Wkj -> hidden node j와 output node k 사이의 weight

- 아키텍쳐: 입력층-은닉층-출력층
- ANN - 결합관계를 나타내는 가중치(w) 도출, 모형 구축
- 입력 신호가 순방향(계층 별)으로 전파되어 출력 계층에 도달하는 feed forward 단계 + 연결 강도(가중치)를 수정하는 오류 역방향 전파 프로세스
- 오차 = 목표 변수와 계산된 값과 관찰된 값의 차이

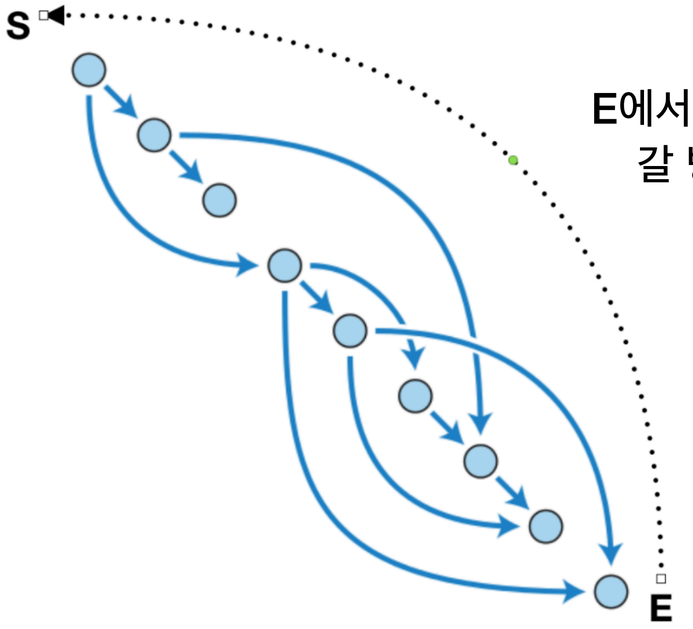


Networks

- Feedforward neural networks -> 네트워크
- **방향성 비순환 그래프(directed acyclic graph)**와 연결
- 비방향 그래프 vs. 방향 그래프 - 방향이 없는 것과 있는 것. 방향그래프는 정점마다 연결된 에지에 화살표로 들어오고 나가는 방향에 대해서 표시를 하는 특성을 가진다.
- 순한 vs 비순환 - 순환: 출발한 노드(정점) v에서 시작하여 끝내 다시 v로 돌아가 순환 반복 될 수 있다.
- E에서 다시 S로 되돌아갈 에지가 없어 순환 반복될 수 없음 -> 비순환 그래프



비방향 그래프 vs. 방향그래프

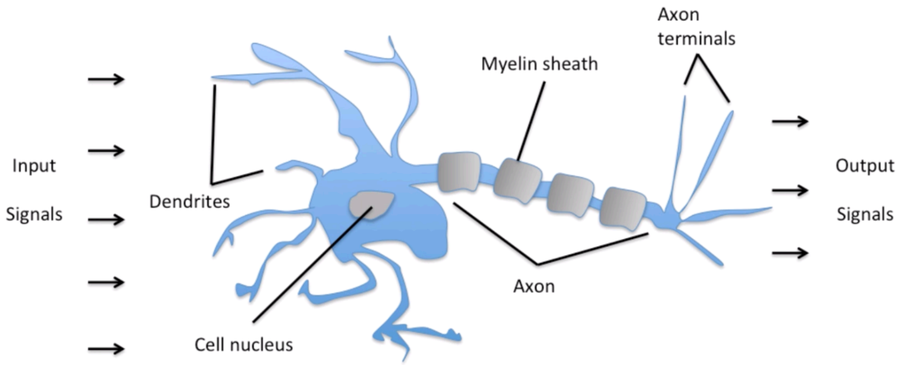


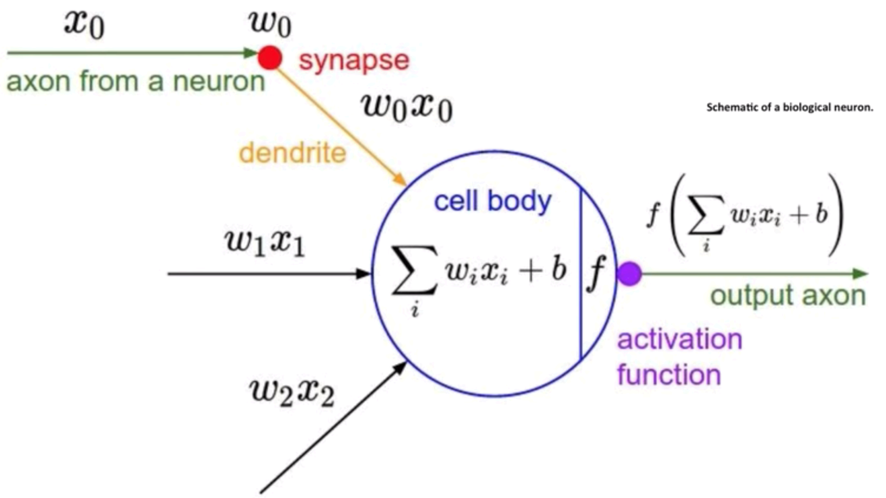
E에서 S로 다시 돌아갈 방법이 없다.

비순환 방향 그래프

Inspired by Neurons

- 신경 과학에서의 영감
- 네트워크의 각 숨겨진 계층은 일반적으로 벡터 값
- 벡터의 각 요소는 뉴런과 유사한 역할로 해석
- 계층이 병렬로 작동하는 여러 단위로 구성되어 있으며 각각 벡터 대 스칼라 함수를 나타내는 것으로 생각
- 단위는 다른 많은 단위로부터 입력을 받고 자체 활성화 값을 계산한다는 점에서 뉴런과 비슷

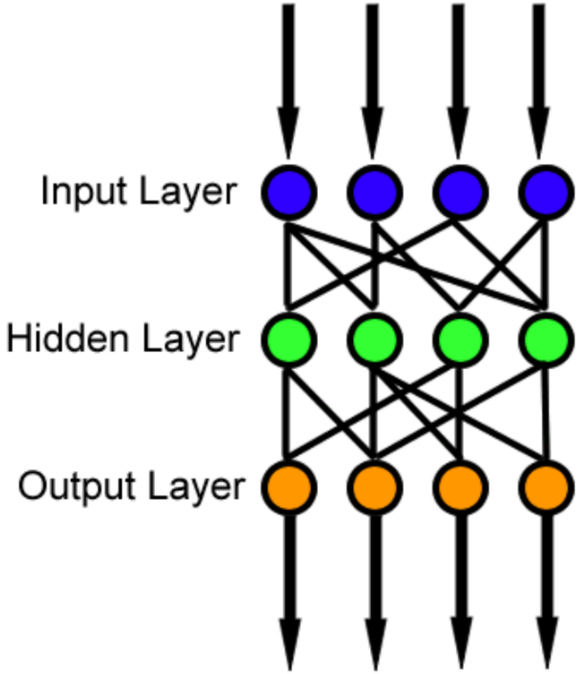




Schematic of a biological neuron.

종류 1. 단층 퍼셉트론

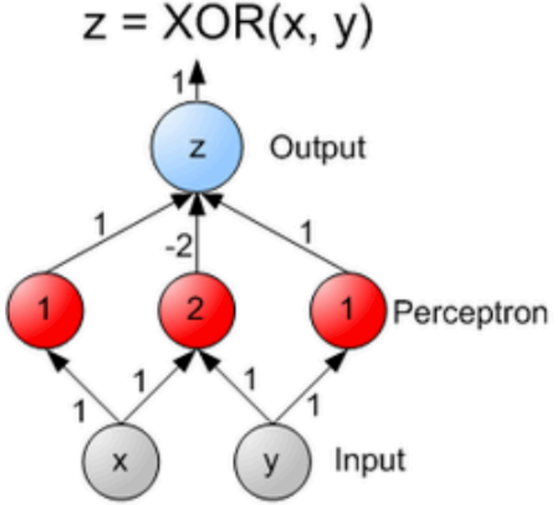
- 가중치 + 입력 값이 각 노드에서 계산. 값이 특정 임계 값 (일반적으로 0)보다 높으면 뉴런이 실행, 활성화 된 값 (일반적으로 1)을 취함. 그렇지 않으면 비활성화 값 (일반적으로 -1) 사용
- 퍼셉트론은 임계 값이 둘 사시에 있는 한, 활성화 및 비활성화 상태에 대한 값을 사용하여 생성 가능.
- 계산된 결과와 샘플 출력 데이터 사이의 오류를 계산하고, 이 값을 사용하여 가중치에 대한 조정을 작성하여 gradient descent 양식 구현.
- 단일 단위 퍼셉트론은 선형으로 분리 가능한 패턴을 학습 할 수 있다.
- Perceptrons - 단층 퍼셉트론 네트워크가 XOR 함수를 배울 수 없음을 보여줌
- 병렬 임계 값 단위 네트워크는 실수 간 간격에서 [-1, 1]까지 모든 연속 함수를 근사시킬 수 있음
- 단일 계층 네트워크 = 로지스틱 회귀 모델 (sigmoid function)
    - Backpropagation



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

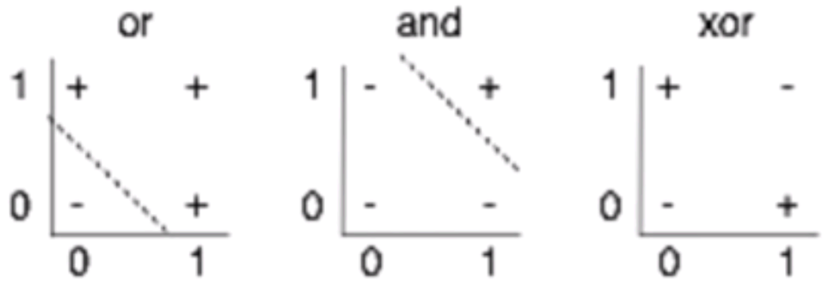
종류 2. 다층 퍼셉트론

- FFNN, 연결된 계산 단위의 여러 계층을 구성
- 한 레이어의 각 뉴런 - 후속 레이어의 뉴런에 연결을 연결
- Sigmoid function 적용



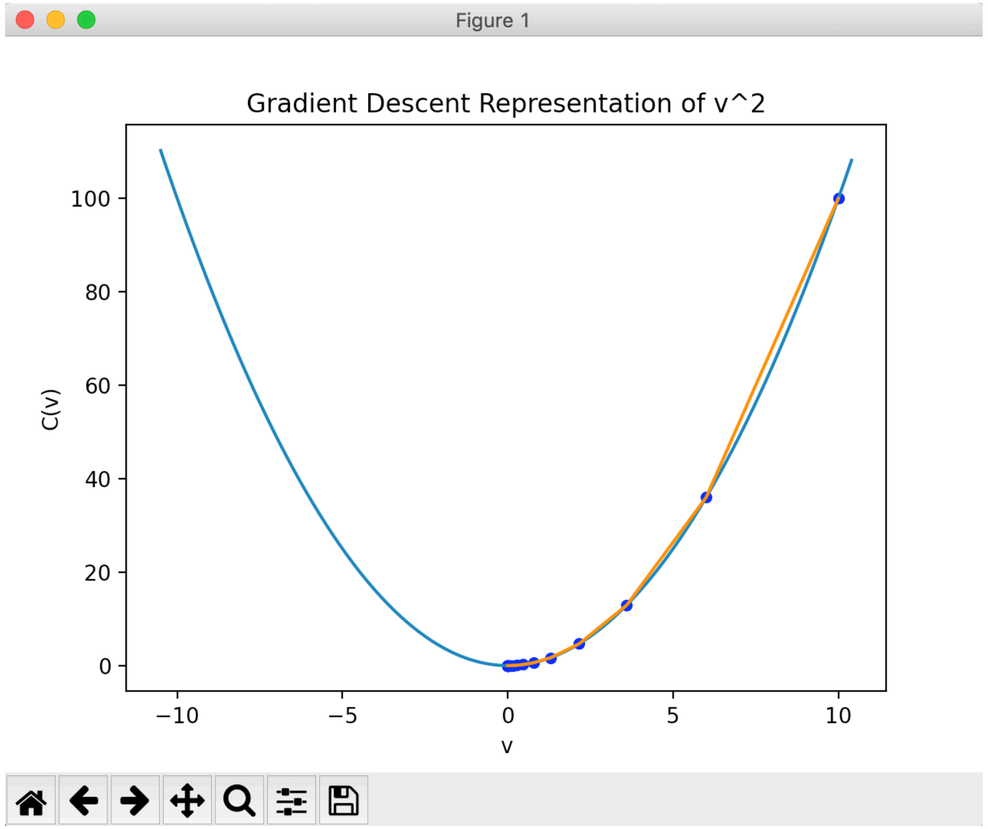
XOR

- Exclusive 하다는 것은 값이 다를때 1이 되고 값이 같으면 0이 된다.
- XOR은 linear한 선을 그을수가 없다. 정확도가 떨어진다.
- Marvin Minsky가 Perceptrons (1969)에서 XOR로는 풀 수가 없다는 것을 수학적으로 증명
- MLP로는 XOR을 풀 수 있다. 그러나 그 안의 weight와 bias를 학습시킬 수 없다.
- backpropagation으로 풀어낸다



Gradient Descent Coding

```python
#implementation of Basic Gradient Descent 1

import matplotlib.pyplot as plt
import numpy as np

def gradient_descent(gradient, start, learn_rate, n_iter=50, tolerance=1e-06):
    vectors = [start]
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)
        if abs(diff) <= tolerance:
            break
        vector += diff
        vectors.append(vector)
    return vectors

vectors = gradient_descent(gradient=lambda v: 2 * v, start=10.0, learn_rate=0.2)
print(vectors)

t = np.arange(-10.5, 10.5, 0.1)
func = lambda x : x ** 2
s = [func(i) for i in t]

vectors = gradient_descent(gradient=lambda v: 2 * v, start=10.0, learn_rate=0.2)
s2 = [func(i) for i in vectors]

plt.plot(t,s,vectors,s2)
plt.scatter(vectors,s2, c='blue', s=20)
plt.xlabel('v')
plt.ylabel('C(v)')
plt.title('Gradient Descent Representation of v^2')
plt.show()
```

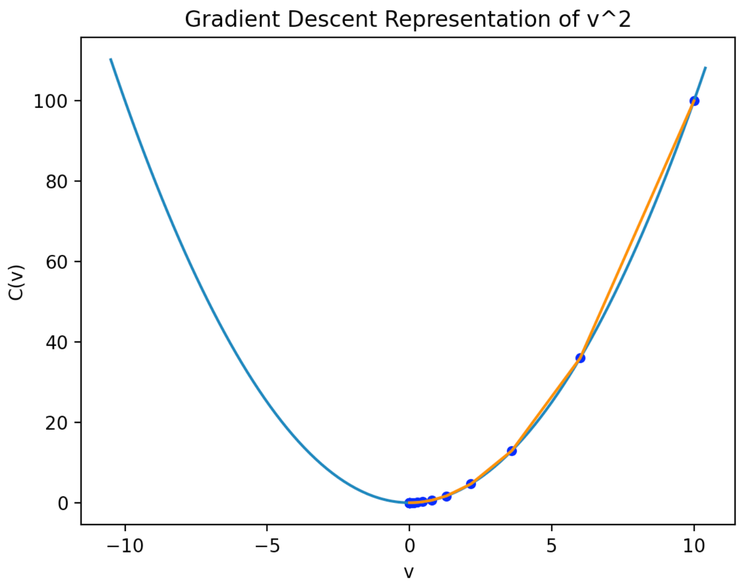



```
========== RESTART: /Users/parkdarlnim/Desktop/혼공/stochastic.py ========
====
[10.0, 6.0, 3.5999999999999996, 2.1599999999999997, 1.2959999999999998, 0
.7775999999999998, 0.46655999999999986, 0.2799359999999999, 0.16796159999
999993, 0.10077695999999996, 0.06046617599999997, 0.036279705599999976, 0
.021767823359999987, 0.013060694015999992, 0.007836416409599995, 0.004701
849845759997, 0.002821109907455998, 0.0016926659444735988, 0.001015599566
6841593, 0.0006093597400104956, 0.0003656158440062973, 0.0002193695064037
784, 0.00013162170384226703, 7.897302230536021e-05, 4.7383813383216124e-0
5, 2.8430288029929674e-05, 1.7058172817957805e-05, 1.0234903690774682e-05
, 6.1409422144648085e-06, 3.684565328678885e-06, 2.210739197207331e-06]
```

Changing Learn Rates

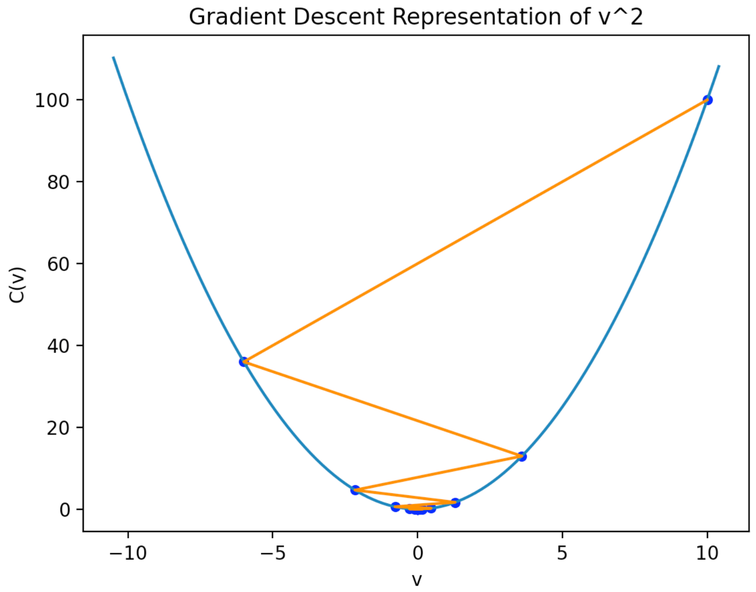
learn_rate = 0.2

2.210739197207331e-06



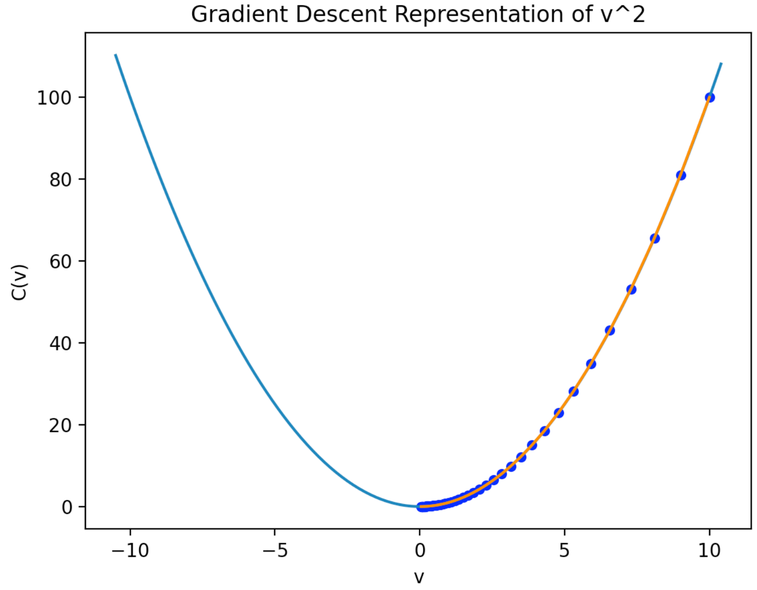
learn_rate = 0.8

-4.775196666596786e-07



learn_rate = 0.05

6.060060671375367



Nonconvex Function Graph

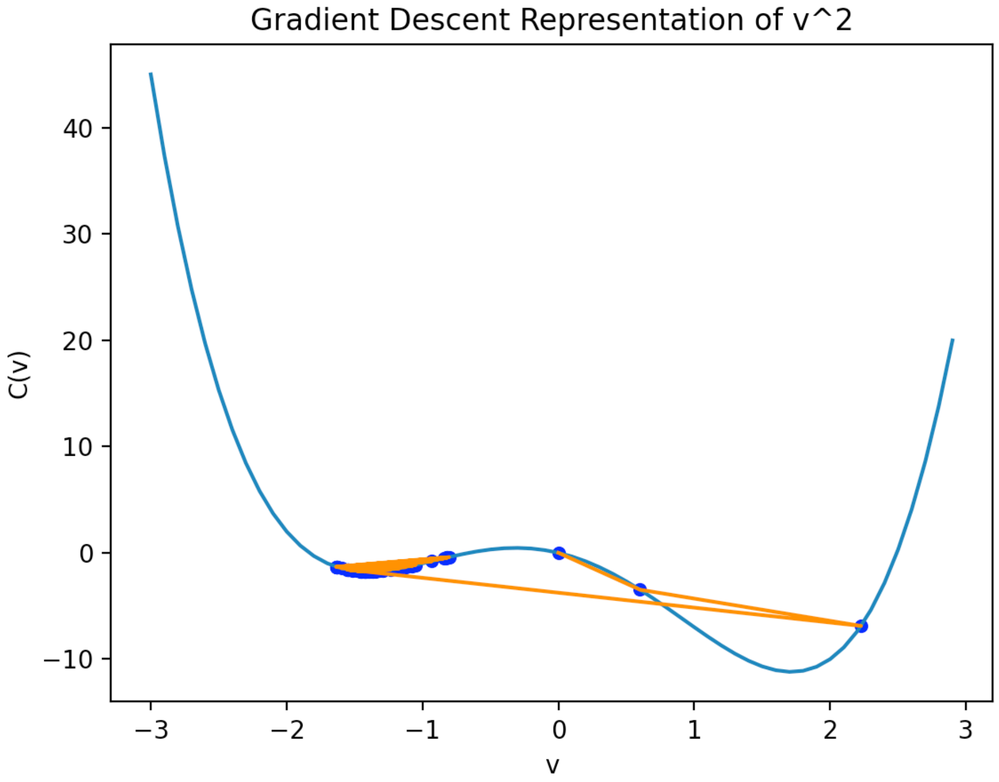
Consider the function $v^4 - 5v^2 - 3v$

Algorithm can get trapped in local minima

Choice of learning rate/starting point can make the difference between finding local min.(-1.42) OR global min.(1.7)

learn_rate = 0.2

-1.4207567437458342



learn_rate = 0.1

1.285401330315467