12주차 과제

7-1. Softmax Regression

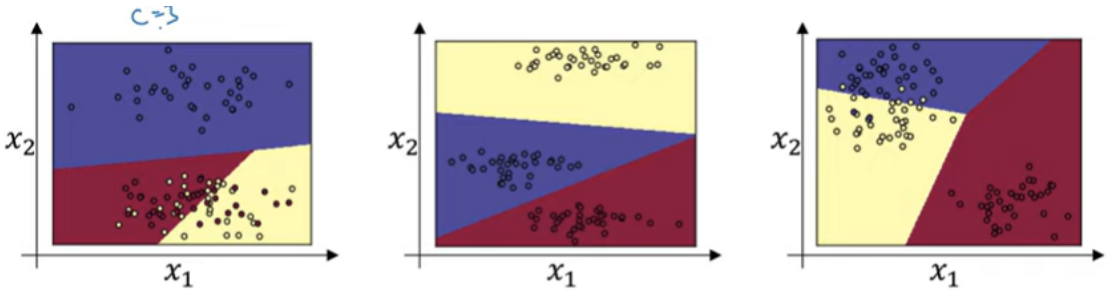
: 분류에서 사용, 로지스틱 회귀를 일반화
→ 로지스틱 회귀완 달리 여러 개의 클래스 분류 시 사용됨.

C = # of classes
→ Classes : {0, 1, 2, ... , (C-1)}
→
$n^{[L]} = C$: 출력층의 노드 수
→
$n_1^{[L]} = P(0|X),\ n_2^{[L]} = P(1|X),\ \cdots, n_C^{[L]} = P(C-1|X)$
: 입력값 X가 주어졌을 때 마지막 층의 각각 노드의 값은 각 클래스에 포함될 확률!
→ 마지막 층은 (C, 1)의 차원을 가지게 됨!
→
$\sum$ 마지막 층 모든 노드의 출력값($\hat{y}$) = 1 ← 이 특징으로 소프트 맥스 정의

$$for\ \ t_i = e^{z_i^{[L]}},\ \ a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{j=1}^{C} t_j} = \frac{t_j}{\sum_{j=1}^{C} t_j}$$

→ if C = 2, 소프트맥스 함수는 로지스틱 회귀와 같은 식, Sigmoid 함수 형태를 갖는다! (그래서 로지스틱 회귀의 일반화된 버전 즉, 이진 분류가 아닌 경우에도 사용 가능한 형태라는 것!)

소프트맥스의 특이한 점은 input, output이 모두 벡터라는 것! (근데 왜 특이한 거지...? 보통은 그냥 실수 값 하나를 받아서 실수 값 하나를 반환하니까...??)



위의 그림처럼 소프트맥스 함수를 사용하면 각 클래스간의 경계가 선형으로 나타나게 됨

7-2. Softmax 분류기 훈련시키기

↔ Hardmax : 벡터에서 가장 큰 값의 인덱스에만 1, 나머지 위치엔 다 0을 반환

[Loss function]

$$L(\hat{y}, y) = -\sum_{j=1}^{C} y_j log\hat{y}_j$$

여기서 y는 0 또는 1($y = I_{\{true\ label==j\}}$ 이런 Identity function으로 생각 가능), y_hat은 확률 값

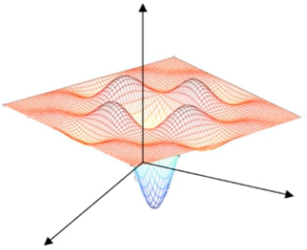
[How to use Gradient Descent with Softmax]

마지막 출력층 (C, 1)의 차원

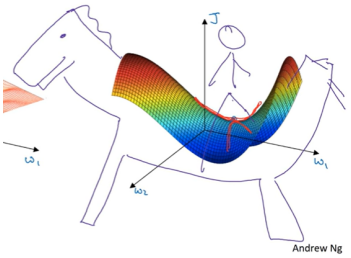
$dz^{[L]} = \hat{y} - y$ 로 역전파 값 구하기 가능

8-1. 지역 최적값 문제 (근데 이제 8장 아니고 4장 내용)

비용함수 최솟값을 구할 때, 전역 비용함수 최솟값을 구하는 게 아닌 지역 최솟값에 빠져버리는 문제가 있다!



비용함수의 경사가 0인 경우 대개 안장점(경사가 0인 지점 중 하나), 측정 지표, 그러니까 어떤 변수를 기준으로 보느냐에 따라 비용함수 값이 최소일수도, 최데일수도 있는 지점!
(옆의 그림도 w1기준으로 보면 최소, w2기준으로 보면 최대값을 가짐!)



→ 높은 차원에선 경사가 0이라고 해서 비용함수가 최솟값을 가지진 않는다!!

충분히 큰 신경망을 학습시킨다면 지역 최적값에 갇힐 일이 잘 없다~!

안정지대 안쪽에서 움직이면 경사가 거의 0에 가깝기 때문에 학습속도가 느려짐 → Adam과 같은 알고리즘으로 학습속도 저하 문제를 해결 가능!

8-2. Tensorflow

```
#TensorFlow version1
import numpy as np
import tensorflow as tf

coefficients = np.array([[1], [-20], [25]])

w = tf.Variable([0], dtype = tf.float32)
#→ 변수 지정(변수의 데이터 타입도 지정 가능)
x = tf.placeholder(tf.float32, [3, 1])
cost = tf.add(tf.add(w**2, tf.multiply(-10, w)), 25)  #(w-5)*
#→ 비용함수 정의
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost
#경사하강법 지정
init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
#전역변수 초기화
sess.run(w)

for i in range(1000):
    session.run(train, feed_dict = {x:coefficients})
```

tensorflow를 이용, → 정방향 전파만 설정해도 자동으로 역전파를 수행 가능한 프레임 워크

Week 12. 발표 정리