12주차

날짜	@2024년 5월 28일
과제	강의 요약 출석 퀴즈
세부내용	[딥러닝 2단계] 7. 다중 클래스 분류 8. 프로그래밍 프레임워크 소개

다중 클래스 분류

1️⃣ Softmax Regression

Recognizing cats, dogs, and baby chicks

- C = 4 : # of classes → (0, 1, 2, 3) : 0~C-1
- 출력층 L의 단위개수 n : $n^{[L]} = C = 4$
- 출력층에서의 출력값 $\hat{y}$은 (4,1)의 벡터
    - P(other|X)
    - P(cat|X)
    - P(dog|X)
    - P(baby chick|X)

    ⇒ 이 확률의 합은 1이 되어야 함

Softmax Layer

- final layer : $z^{[L]} = w^{[L]}a^{[L-1]} + b^{[L]}$
- activation function :
    - 임시변수 : $t = e^{z^{[L]}}$ (4,1)
    - $a^{[L]} = \frac{exp(z^{[L]})}{\sum_{j=1}^{4} t_i}$ (4,1) → $a_i^{[L]} = \frac{t_i}{\sum t_i}$ ⇒ 합이 1이 되도록 함
- Softmax Function g : 입력값과 출력값이 모두 벡터

Softmax examples

- 은닉층이 없는 네트워크에서 소프트맥스가 하는 일
- C=3의 클래스를 갖는 data → 소프트맥스 함수를 학습 → 클래스를 분류
- 은닉층이 없을 때 클래스 사이의 경계는 선형

2️⃣ Softmax 분류기 훈련시키기

Understanding Softmax

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix},\ t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$
$$g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} = a^{[L]}$$

- z의 첫번째 원소가 가장 큰 값을 가짐 → 가장 큰 확률값을 가짐
- “hardmax” : z 중 가장 큰 값이 있는 곳에 1을, 나머지는 0을 갖는 벡터로 대응시키는 방법
- Softmax Regression generalizes logistic regression to C classes
    - if C=2, softmax = logistic regression

Loss Function

- $y = [0, 1, 0, 0]^T$ → cat
- $a^{[L]} = \hat{y} = [0.3, 0.2, 0.1, 0.4]^T$
- $L(\hat{y}, y) = -\sum_{j=1}^{C} y_j \log \hat{y}_j$ → 최소화 ⇒ $\hat{y}_j$를 최대화
    - 클래스에 대응하는 확률을 최대화하는 것 → likelihood와 유사

$$Y = [y^{(1)}, \cdots, y^{(m)}]^T = \begin{bmatrix} 0 & 0 & 1 & \cdots \\ 1 & 0 & 0 & \cdots \\ 0 & 1 & 0 & \cdots \\ 0 & 0 & 0 & \cdots \end{bmatrix}$$
$$\hat{Y} = [\hat{y}^{(1)}, \cdots, \hat{y}^{(m)}]^T = \begin{bmatrix} 0.3 & \cdots \\ 0.2 & \cdots \\ 0.1 & \cdots \\ 0.4 & \cdots \end{bmatrix}$$

- Y와 $\hat{Y}$ 모두 (4,m) 차원

‘

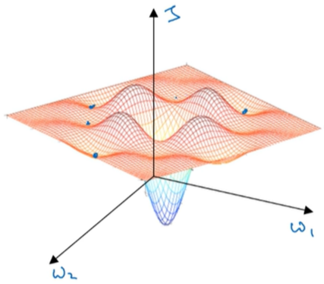
Gradient Descent with Softmax

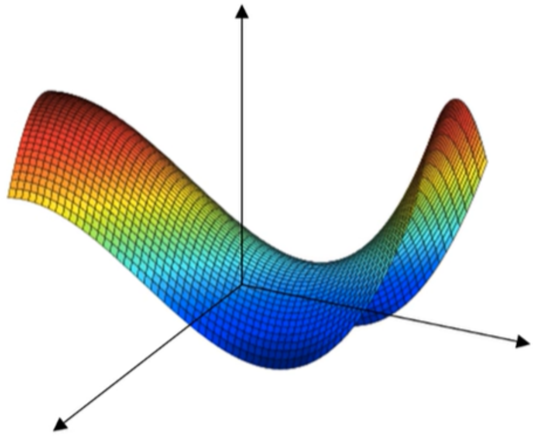
- backprop : $\frac{\partial J}{\partial z^{[L]}} = dz^{[L]} = \hat{y} - y$

프로그래밍 프레임워크 소개

1️⃣ 지역 최적값 문제

Local Optima in Neural Networks





- 전역 최적값에 도달하기 전, 지역 최적값에 갇혀버릴 수 있음
- 고차원 비용함수에서 경사가 0 → 안장점 (not 지역 최적값)

Problem of Plateaus

- 안정지대 : 안장점으로 향하는 구간, 미분값이 오랫동안 0에 가깝게 유지됨 → 학습이 느려짐
    - Adam, RMSprop 등 알고리즘으로 해결 가능
- 충분히 큰 신경망을 학습시키면 지역 최적값에 갇히는 일 거의 없음

2️⃣ Tensorflow

Motivating Problem

- 비용함수 : $J(w) = w^2 - 10w + 25 = (w-5)^2$ 를 텐서플로우로 최소화하기

```
import numpy as np
import tensorflow as tf

coefficient = np.array([[1.],[-10.],[25.]])

w = tf.Variable(0,dtype=tf.float32)  # w값 0으로 초기화
X = tf.placeholder(tf.float32, [3,1])  # (3x1) 형태 데이터
# cost = tf.add(tf.add(w**2, tf.multiply(-10.,w)),25)
cost = w**2 - 10*w + 25  # cost function 정의
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]  # 이차함수의 계수를 조절
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)   # 경사하강법 학습 알고리

init = tf.global_varaibles_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))  # 학습 실행
# 0.0

session.run(train)  # 1-step
print(session.run(w))
# 0.1
```

```
for i in range(1000):
    session.run(train)
print(session.run(w))
# 4.99999


coefficient = np.array([[1.],[-20.],[100.]])
session.run(train, feed_dict={x:coefficients})  # 1-step
print(session.run(w))
# 0.2

for i in range(1000):
    session.run(train, feed_dict={x:coefficients})
print(session.run(w))
# 9.99998
```